Horus C++ Reference

Version 2.0 - Jan 2003

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

1 Class Overview **2**

1.1 Class overview 2

2 Pixels **3**

2.1 Pixels 3

2.2 Pixel value representation 3

2.3 Arithmetic data types 3

2.4 Unary operations on pixel values 4

2.5 Binary operations on pixel values 5

2.6 Color operations on pixel values 6

3 Images **8**

3.1 Images 8

3.2 Image definition and generic operations 8

3.3 Images and semantics 8

3.4 Image definition 9

3.5 Generic image operations 10

3.6 Global image functions 10

3.7 Image data representation 12

3.8 The way image operations work 15

4 Detector **65**

4.1 Detector 65

4.2 Histogram 65

4.3 Segmentation 65

5 Geometry **67**

5.1 BSplines 67

6 Registry **72**

6.1 Registry 72

7 Global Image Functions Reference **75**

7.1 HxAbs.h File Reference 75
7.2 HxAcos.h File Reference 76
7.3 HxAdd.h File Reference 77
7.4 HxAddBinaryNoise.h File Reference 78
7.5 HxAddGaussianNoise.h File Reference 79
7.6 HxAddPoissonNoise.h File Reference 80
7.7 HxAddSat.h File Reference 80
7.8 HxAddUniformNoise.h File Reference 81
7.9 HxAddVal.h File Reference 82
7.10 HxAffinePix.h File Reference 83
7.11 HxAlternateSequentialFilter.h File Reference 83
7.12 HxAnd.h File Reference 86
7.13 HxAndVal.h File Reference 87
7.14 HxAreaClosing.h File Reference 88
7.15 HxAreaOpening.h File Reference 88
7.16 HxArg.h File Reference 89
7.17 HxAsin.h File Reference 89
7.18 HxAtan.h File Reference 91
7.19 HxAtan2.h File Reference 91
7.20 HxBlob2dList.h File Reference 92
7.21 HxByte.h File Reference 93
7.22 HxCannyEdgeMap.h File Reference 93
7.23 HxCannyThreshold.h File Reference 94
7.24 HxCannyThresholdAlt.h File Reference 95
7.25 HxCannyThresholdRec.h File Reference 96
7.26 HxCeil.h File Reference 97
7.27 HxClosing.h File Reference 98
7.28 HxClosingByReconstruction.h File Reference 98
7.29 HxClosingByReconstructionTopHat.h File Reference 99
7.30 HxClosingTopHat.h File Reference 99
7.31 HxColConvert.h File Reference 100
7.32 HxColor.h File Reference 106
7.33 HxColorSpace.h File Reference 107
7.34 HxComplement.h File Reference 108

7.35 HxConditionalDilation.h File Reference . . . 109
7.36 HxConditionalErosion.h File Reference . . . 110
7.37 HxConjugate.h File Reference . . . 111
7.38 HxContrastStretch.h File Reference . . . 112
7.39 HxConvGauss2d.h File Reference . . . 112
7.40 HxConvGauss3d.h File Reference . . . 114
7.41 HxConvKernelSeparated.h File Reference . . . 116
7.42 HxConvKernelSeparated2d.h File Reference . . . 117
7.43 HxConvolution.h File Reference . . . 117
7.44 HxCos.h File Reference . . . 118
7.45 HxCosh.h File Reference . . . 119
7.46 HxCross.h File Reference . . . 120
7.47 HxCrossVal.h File Reference . . . 121
7.48 HxDefuz.h File Reference . . . 122
7.49 HxDilation.h File Reference . . . 123
7.50 HxDisplayOF.h File Reference . . . 123
7.51 HxDistanceTransform.h File Reference . . . 124
7.52 HxDistanceTransformMM.h File Reference . . . 125
7.53 HxDiv.h File Reference . . . 126
7.54 HxDivVal.h File Reference . . . 127
7.55 HxDot.h File Reference . . . 130
7.56 HxDotVal.h File Reference . . . 131
7.57 HxEqual.h File Reference . . . 132
7.58 HxEqualVal.h File Reference . . . 134
7.59 HxErosion.h File Reference . . . 135
7.60 HxExp.h File Reference . . . 135
7.61 HxExportByteData.h File Reference . . . 135
7.62 HxExportDoubleData.h File Reference . . . 136
7.63 HxExportFloatData.h File Reference . . . 137
7.64 HxExportIntData.h File Reference . . . 138
7.65 HxExportMatlabPixels.h File Reference . . . 139
7.66 HxExportPpmPixels.h File Reference . . . 139
7.67 HxExportShortData.h File Reference . . . 140
7.68 HxExtend.h File Reference . . . 140
7.69 HxExtendVal.h File Reference . . . 142
7.70 HxFloor.h File Reference . . . 143

7.71 HxFuncBorderOp.h File Reference 144
7.72 HxFuncBpo.c File Reference 152
7.73 HxFuncExportExtra.c File Reference 153
7.74 HxFuncGenConv2d.c File Reference 156
7.75 HxFuncGenConv2dK1d.c File Reference 157
7.76 HxFuncGenConv2dSep.c File Reference 164
7.77 HxFuncGenConv3d.c File Reference 181
7.78 HxFuncGenConv3dK1d.c File Reference 181
7.79 HxFuncInOut.c File Reference 185
7.80 HxFuncInOutInit.h File Reference 190
7.81 HxFuncKernelNgbOp2d.c File Reference 193
7.82 HxFuncMNpo.c File Reference 196
7.83 HxFuncMpo.c File Reference 197
7.84 HxFuncNgbOp2d.c File Reference 198
7.85 HxFuncNgbOp2dExtra.c File Reference 202
7.86 HxFuncNgbOp2dExtra2.c File Reference 206
7.87 HxFuncRecGenConv2d.c File Reference 211
7.88 HxFuncRecGenConv2dK1d.c File Reference 213
7.89 HxFuncRgbOp2d.c File Reference 219
7.90 HxFuncRgbOp3d.c File Reference 220
7.91 HxFuncSet.c File Reference 222
7.92 HxFuncUpo.c File Reference 224
7.93 HxGauss.h File Reference 225
7.94 HxGaussDerivative2d.h File Reference 226
7.95 HxGaussDerivative3d.h File Reference 227
7.96 HxGaussianDeblur.h File Reference 229
7.97 HxGeodesicDistanceTransform.h File Reference 230
7.98 HxGeoIntType.h File Reference 230
7.99 HxGetBlobFeatures.h File Reference 231
7.100HxGetPoints.h File Reference 232
7.101HxGetValues.h File Reference 232
7.102HxGreaterEqual.h File Reference 232
7.103HxGreaterEqualVal.h File Reference 233
7.104HxGreaterThan.h File Reference 234
7.105HxGreaterThanVal.h File Reference 235
7.106HxHighlightRegion.h File Reference 236

7.107HxHilditchSkeleton.h File Reference . . . 237
7.108HxHitOrMiss.h File Reference . . . 238
7.109HxIdentMaskCentralMoments.h File Reference . . . 238
7.110HxIdentMaskMean.h File Reference . . . 239
7.111HxIdentMaskMedian.h File Reference . . . 240
7.112HxIdentMaskMoments.h File Reference . . . 240
7.113HxIdentMaskStDev.h File Reference . . . 241
7.114HxIdentMaskSum.h File Reference . . . 242
7.115HxIdentMaskVariance.h File Reference . . . 243
7.116HxImageAsByte.h File Reference . . . 244
7.117HxImageAsComplex.h File Reference . . . 244
7.118HxImageAsDouble.h File Reference . . . 245
7.119HxImageAsFloat.h File Reference . . . 246
7.120HxImageAsInt.h File Reference . . . 247
7.121HxImageAsShort.h File Reference . . . 247
7.122HxImageAsVec2Byte.h File Reference . . . 248
7.123HxImageAsVec2Double.h File Reference . . . 249
7.124HxImageAsVec2Float.h File Reference . . . 250
7.125HxImageAsVec2Int.h File Reference . . . 250
7.126HxImageAsVec2Short.h File Reference . . . 251
7.127HxImageAsVec3Byte.h File Reference . . . 252
7.128HxImageAsVec3Double.h File Reference . . . 253
7.129HxImageAsVec3Float.h File Reference . . . 253
7.130HxImageAsVec3Int.h File Reference . . . 254
7.131HxImageAsVec3Short.h File Reference . . . 255
7.132HxImageMaxSize.h File Reference . . . 256
7.133HxImageMinSize.h File Reference . . . 256
7.134HxImagesFromFile.h File Reference . . . 257
7.135HxImagesToFile.h File Reference . . . 258
7.136HxImageToSegmentation.h File Reference . . . 258
7.137HxInf.h File Reference . . . 259
7.138HxInfimumReconstruction.h File Reference . . . 260
7.139HxInfVal.h File Reference . . . 261
7.140HxInverseProjectRange.h File Reference . . . 262
7.141HxKuwahara.h File Reference . . . 263
7.142HxLabel.h File Reference . . . 264

7.143HxLabel2.h File Reference 265
7.144HxLeftShift.h File Reference 265
7.145HxLeftShiftVal.h File Reference 267
7.146HxLessEqual.h File Reference 268
7.147HxLessEqualVal.h File Reference 269
7.148HxLessThan.h File Reference 270
7.149HxLessThanVal.h File Reference 271
7.150HxLocalMode.h File Reference 272
7.151HxLog.h File Reference 272
7.152HxLog10.h File Reference 273
7.153HxLUT.h File Reference 274
7.154HxMakeFrom2Images.h File Reference 275
7.155HxMakeFrom3Images.h File Reference 276
7.156HxMakeFromByteData.h File Reference 278
7.157HxMakeFromDoubleData.h File Reference 279
7.158HxMakeFromFile.h File Reference 280
7.159HxMakeFromFloatData.h File Reference 280
7.160HxMakeFromGenerator.h File Reference 281
7.161HxMakeFromGrayValue.h File Reference 282
7.162HxMakeFromImage.h File Reference 283
7.163HxMakeFromImport.h File Reference 284
7.164HxMakeFromIntData.h File Reference 284
7.165HxMakeFromJavaRgb.h File Reference 285
7.166HxMakeFromMatlab.h File Reference 286
7.167HxMakeFromNamedGenerator.h File Reference 287
7.168HxMakeFromPpmPixels.h File Reference 288
7.169HxMakeFromShortData.h File Reference 288
7.170HxMakeFromSignature.h File Reference 289
7.171HxMakeFromValue.h File Reference 290
7.172HxMakeGaussian1d.h File Reference 291
7.173HxMakeParabola1d.h File Reference 292
7.174HxMatrixConv.h File Reference 293
7.175HxMax.h File Reference 293
7.176HxMaxVal.h File Reference 295
7.177HxMin.h File Reference 296
7.178HxMinVal.h File Reference 297

7.179HxMod.h File Reference 298
7.180HxModVal.h File Reference 299
7.181HxMorphologicalContour.h File Reference 300
7.182HxMorphologicalGradient.h File Reference 300
7.183HxMorphologicalGradient2.h File Reference 301
7.184HxMul.h File Reference 301
7.185HxMulVal.h File Reference 303
7.186HxNegate.h File Reference 304
7.187HxNonMaxSuppressionGradDir.h File Reference 305
7.188HxNorm1.h File Reference 305
7.189HxNorm2.h File Reference 306
7.190HxNorm2Sqr.h File Reference 306
7.191HxNormalizedCorrelation.h File Reference 307
7.192HxNormInf.h File Reference 308
7.193HxNotEqual.h File Reference 308
7.194HxNotEqualVal.h File Reference 310
7.195HxOpening.h File Reference 311
7.196HxOpeningByReconstruction.h File Reference 311
7.197HxOpeningByReconstructionTopHat.h File Reference 311
7.198HxOpeningTopHat.h File Reference 312
7.199HxOpponentColor.h File Reference 312
7.200HxOpticalFlow.h File Reference 312
7.201HxOpticalFlowMultiScale.h File Reference 313
7.202HxOr.h File Reference 315
7.203HxOrVal.h File Reference 316
7.204HxParabolicDilation.h File Reference 317
7.205HxParabolicErosion.h File Reference 318
7.206HxPeakRemoval.h File Reference 319
7.207HxPercentile.h File Reference 319
7.208HxPixInf.h File Reference 320
7.209HxPixMax.h File Reference 321
7.210HxPixMin.h File Reference 322
7.211HxPixProduct.h File Reference 323
7.212HxPixSum.h File Reference 323
7.213HxPixSup.h File Reference 324
7.214HxPoint.h File Reference 325

7.215HxPointInt.h File Reference 325
7.216HxPointList.h File Reference 326
7.217HxPow.h File Reference 326
7.218HxPowVal.h File Reference 328
7.219HxProjectRange.h File Reference 329
7.220HxRecGauss.h File Reference 330
7.221HxReciprocal.h File Reference 331
7.222HxReflect.h File Reference 332
7.223HxRegionalMaxima.h File Reference 332
7.224HxRegionalMinima.h File Reference 334
7.225HxRegKeyList.h File Reference 335
7.226HxRegValueList.h File Reference 336
7.227HxRestrict.h File Reference 337
7.228HxRGB2Intensity.h File Reference 339
7.229HxRightShift.h File Reference 339
7.230HxRightShiftVal.h File Reference 341
7.231HxRotate.h File Reference 342
7.232HxRound.h File Reference 343
7.233HxScale.h File Reference 343
7.234HxSegmentationCentralMoments.h File Reference 344
7.235HxSegmentationHistogram.h File Reference 345
7.236HxSegmentationMean.h File Reference 345
7.237HxSegmentationMedian.h File Reference 346
7.238HxSegmentationMoments.h File Reference 346
7.239HxSegmentationStDev.h File Reference 347
7.240HxSegmentationSum.h File Reference 348
7.241HxSegmentationVariance.h File Reference 348
7.242HxSetBorderValue.h File Reference 349
7.243HxSetPartImage.c File Reference 349
7.244HxSetPartImage.h File Reference 350
7.245HxSin.h File Reference 351
7.246HxSinh.h File Reference 351
7.247HxSizes.h File Reference 352
7.248HxSkeleton.h File Reference 353
7.249HxSKIZ.h File Reference 353
7.250HxSqrt.h File Reference 354

7.251HxSquaredDistance.h File Reference 355
7.252HxStringList.h File Reference 357
7.253HxStringNative.h File Reference 358
7.254HxSub.h File Reference 358
7.255HxSubSat.h File Reference 360
7.256HxSubVal.h File Reference 361
7.257HxSup.h File Reference 362
7.258HxSupremumReconstruction.h File Reference 363
7.259HxSupVal.h File Reference 364
7.260HxTagList.h File Reference 365
7.261HxTan.h File Reference 367
7.262HxTanh.h File Reference 367
7.263HxThickening.h File Reference 368
7.264HxThinning.h File Reference 369
7.265HxThreshold.h File Reference 370
7.266HxTranslate.h File Reference 371
7.267HxTranspose.c File Reference 371
7.268HxTranspose.h File Reference 373
7.269HxTriStateThreshold.h File Reference 373
7.270HxUnaryMax.h File Reference 374
7.271HxUnaryMin.h File Reference 375
7.272HxUnaryProduct.h File Reference 376
7.273HxUnarySum.h File Reference 376
7.274HxUniform.h File Reference 377
7.275HxUniformNonSep.h File Reference 378
7.276HxUpoSetPartImageInst.c File Reference 378
7.277HxValleyRemoval.h File Reference 379
7.278HxValueList.h File Reference 379
7.279HxValueType.h File Reference 380
7.280HxVec2Byte.h File Reference 381
7.281HxVec2Float.h File Reference 381
7.282HxVec2Short.h File Reference 382
7.283HxVec3Byte.h File Reference 382
7.284HxVec3Float.h File Reference 383
7.285HxVec3Short.h File Reference 383
7.286HxWatershed.h File Reference 384

7.287HxWatershedMarkers.h File Reference . . . 385
7.288HxWatershedMarkers2.h File Reference . . . 386
7.289HxWatershedSlow.h File Reference . . . 387
7.290HxWeightMaskSum.h File Reference . . . 394
7.291HxWriteFile.h File Reference . . . 394
7.292HxXor.h File Reference . . . 396
7.293HxXorVal.h File Reference . . . 397

8 Class Reference **399**

8.1 DeviceEnumerator Class Reference . . . 399
8.2 HxArrowR2 Class Reference . . . 400
8.3 HxBlob2d Class Reference . . . 402
8.4 HxBlob2dFeature Class Reference . . . 406
8.5 HxBlob2dFeatureTem Class Template Reference . . . 408
8.6 HxBlob2dList Class Reference . . . 410
8.7 HxBlob2dPtrLess Struct Reference . . . 410
8.8 HxBlob2dRelation Class Reference . . . 411
8.9 HxBoundingBox Class Reference . . . 414
8.10 HxBpoAdd Class Template Reference . . . 417
8.11 HxBpoAddAssign Struct Template Reference . . . 419
8.12 HxBpoAddSat Class Template Reference . . . 420
8.13 HxBpoAnd Class Template Reference . . . 422
8.14 HxBpoBind2Val Class Template Reference . . . 424
8.15 HxBpoCross Class Template Reference . . . 425
8.16 HxBpoDiv Class Template Reference . . . 426
8.17 HxBpoDot Class Template Reference . . . 428
8.18 HxBpoEqual Class Template Reference . . . 429
8.19 HxBpoGreaterEqual Class Template Reference . . . 431
8.20 HxBpoGreaterThan Class Template Reference . . . 432
8.21 HxBpoHighlightRegion Class Template Reference . . . 434
8.22 HxBpoInf Class Template Reference . . . 436
8.23 HxBpoInfAssign Struct Template Reference . . . 437
8.24 HxBpoLeftShift Class Template Reference . . . 439
8.25 HxBpoLessEqual Class Template Reference . . . 440
8.26 HxBpoLessThan Class Template Reference . . . 441
8.27 HxBpoMax Class Template Reference . . . 443
8.28 HxBpoMaxAssign Struct Template Reference . . . 444

8.29 HxBpoMin Class Template Reference 446
8.30 HxBpoMinAssign Struct Template Reference 447
8.31 HxBpoMod Class Template Reference 449
8.32 HxBpoMul Class Template Reference 450
8.33 HxBpoMulAssign Struct Template Reference 452
8.34 HxBpoNotEqual Class Template Reference 453
8.35 HxBpoOr Class Template Reference 455
8.36 HxBpoPow Class Template Reference 456
8.37 HxBpoRightShift Class Template Reference 458
8.38 HxBpoSqrDst Class Template Reference 459
8.39 HxBpoSub Class Template Reference 461
8.40 HxBpoSubAssign Struct Template Reference 462
8.41 HxBpoSubSat Class Template Reference 464
8.42 HxBpoSup Class Template Reference 466
8.43 HxBpoSupAssign Struct Template Reference 467
8.44 HxBpoXor Class Template Reference 469
8.45 HxBSplineBasis Class Reference 470
8.46 HxBSplineCurve Class Reference 477
8.47 HxBSplineInterval Class Reference 490
8.48 HxClassName Struct Template Reference 495
8.49 HxCnum Class Reference 496
8.50 HxColor Class Reference 498
8.51 HxComplex Class Reference 506
8.52 HxCoord Struct Reference 526
8.53 HxDataPtr2dScalarTem Class Template Reference 526
8.54 HxDataPtr2dTem Class Template Reference 527
8.55 HxDataPtr3dScalarTem Class Template Reference 528
8.56 HxDiyTranspose Class Template Reference 529
8.57 HxExportExtraIdentMaskCentralMoments Class Template Reference 530
8.58 HxExportExtraIdentMaskMean Class Template Reference 534
8.59 HxExportExtraIdentMaskMedian Class Template Reference 536
8.60 HxExportExtraIdentMaskMoments Class Template Reference 538
8.61 HxExportExtraIdentMaskStdev Class Template Reference 541
8.62 HxExportExtraIdentMaskSum Class Template Reference 543
8.63 HxExportExtraWeightMaskSum Class Template Reference 546
8.64 HxHistogram Class Reference 548

8.65 HxIfRbPair Class Reference 580
8.66 HxImageData Class Reference 581
8.67 HxImageFactory Class Reference 605
8.68 HxImageList Class Reference 617
8.69 HxImageRep Class Reference 620
8.70 HxImageSeq Class Reference 646
8.71 HxImageSeqData Class Reference 653
8.72 HxImageSeqDXMedia Class Reference 657
8.73 HxImageSeqIter Class Reference 661
8.74 HxImageSeqMDC Class Reference 665
8.75 HxImageSig2dByte Class Reference 669
8.76 HxImageSig2dComplex Class Reference 670
8.77 HxImageSig2dDouble Class Reference 671
8.78 HxImageSig2dFloat Class Reference 672
8.79 HxImageSig2dInt Class Reference 672
8.80 HxImageSig2dShort Class Reference 673
8.81 HxImageSig2dVec2Byte Class Reference 674
8.82 HxImageSig2dVec2Double Class Reference 675
8.83 HxImageSig2dVec2Float Class Reference 675
8.84 HxImageSig2dVec2Int Class Reference 676
8.85 HxImageSig2dVec2Short Class Reference 677
8.86 HxImageSig2dVec3Byte Class Reference 678
8.87 HxImageSig2dVec3Double Class Reference 678
8.88 HxImageSig2dVec3Float Class Reference 679
8.89 HxImageSig2dVec3Int Class Reference 680
8.90 HxImageSig2dVec3Short Class Reference 681
8.91 HxImageSig3dByte Class Reference 681
8.92 HxImageSig3dDouble Class Reference 682
8.93 HxImageSig3dFloat Class Reference 683
8.94 HxImageSig3dInt Class Reference 684
8.95 HxImageSig3dShort Class Reference 684
8.96 HxImageSignature Class Reference 685
8.97 HxImageTem Class Template Reference 693
8.98 HxImageTem2d Class Template Reference 697
8.99 HxImageTem3d Class Template Reference 699
8.100HxImgFtorBpo Class Template Reference 700

8.101HxImgFtorBpoKey Class Reference 703
8.102HxImgFtorDescription Class Reference 704
8.103HxImgFtorDiy Class Template Reference 705
8.104HxImgFtorDiyKey Class Reference 707
8.105HxImgFtorExportExtra Class Template Reference 708
8.106HxImgFtorExportExtraKey Class Reference 710
8.107HxImgFtorGenConv2d Class Template Reference 711
8.108HxImgFtorGenConv2dK1d Class Template Reference 714
8.109HxImgFtorGenConv2dSep Class Template Reference 717
8.110HxImgFtorGenConv2dSepKey Class Reference 720
8.111HxImgFtorGenConv3d Class Template Reference 721
8.112HxImgFtorGenConv3dK1d Class Template Reference 723
8.113HxImgFtorGenConvK1dKey Class Reference 726
8.114HxImgFtorGenConvKey Class Reference 727
8.115HxImgFtorI1 Class Reference 728
8.116HxImgFtorI1Cast Class Template Reference 731
8.117HxImgFtorI1CastKey Class Reference 736
8.118HxImgFtorI1Key Class Reference 736
8.119HxImgFtorI2 Class Reference 737
8.120HxImgFtorI2Cast Class Template Reference 743
8.121HxImgFtorI2CastKey Class Reference 754
8.122HxImgFtorI2Key Class Reference 755
8.123HxImgFtorI3 Class Reference 756
8.124HxImgFtorI3Cast Class Template Reference 759
8.125HxImgFtorI3CastKey Class Reference 767
8.126HxImgFtorI3Key Class Reference 768
8.127HxImgFtorI4 Class Reference 770
8.128HxImgFtorI4Cast Class Template Reference 772
8.129HxImgFtorI4CastKey Class Reference 776
8.130HxImgFtorI4Key Class Reference 777
8.131HxImgFtorIM Class Reference 778
8.132HxImgFtorIMCast Class Template Reference 779
8.133HxImgFtorIMCastKey Class Reference 783
8.134HxImgFtorIMKey Class Reference 784
8.135HxImgFtorIMN Class Reference 784
8.136HxImgFtorIMNCast Class Template Reference 786

8.137HxImgFtorIMNCastKey Class Reference 789
8.138HxImgFtorIMNKey Class Reference 790
8.139HxImgFtorInOut Class Template Reference 791
8.140HxImgFtorInOutKey Class Reference 794
8.141HxImgFtorKernelNgb2d Class Template Reference 795
8.142HxImgFtorKernelNgbKey Class Reference 798
8.143HxImgFtorKey Class Reference 799
8.144HxImgFtorKeyNameTable Class Reference 803
8.145HxImgFtorMNpo Class Template Reference 804
8.146HxImgFtorMNpoKey Class Reference 806
8.147HxImgFtorMpo Class Template Reference 807
8.148HxImgFtorMpoKey Class Reference 809
8.149HxImgFtorNgb2d Class Template Reference 810
8.150HxImgFtorNgb2dExtra Class Template Reference 813
8.151HxImgFtorNgb2dExtra2 Class Template Reference 816
8.152HxImgFtorNgbExtra2Key Class Reference 819
8.153HxImgFtorNgbExtraKey Class Reference 820
8.154HxImgFtorNgbKey Class Reference 821
8.155HxImgFtorObserver Class Reference 822
8.156HxImgFtorQueueBased Class Template Reference 822
8.157HxImgFtorQueueBasedKey Class Reference 827
8.158HxImgFtorRecGenConv2d Class Template Reference 828
8.159HxImgFtorRecGenConv2dK1d Class Template Reference 830
8.160HxImgFtorRecGenConvK1dKey Class Reference 833
8.161HxImgFtorRecGenConvKey Class Reference 834
8.162HxImgFtorRgb2d Class Template Reference 835
8.163HxImgFtorRgb3d Class Template Reference 837
8.164HxImgFtorRgbKey Class Reference 840
8.165HxImgFtorRuleBase Class Reference 840
8.166HxImgFtorSet Class Template Reference 847
8.167HxImgFtorSetBorder2d Class Template Reference 849
8.168HxImgFtorSetBorder3d Class Template Reference 852
8.169HxImgFtorSetBorderKey Class Reference 854
8.170HxImgFtorSetKey Class Reference 855
8.171HxImgFtorTable Class Reference 856
8.172HxImgFtorTableTem Class Template Reference 858

8.173HxImgFtorUpo Class Template Reference 859
8.174HxImgFtorUpoKey Class Reference 861
8.175HxImgFunctor Class Reference 862
8.176HxInstantiatorAbs Class Template Reference 864
8.177HxInstantiatorAcos Class Template Reference 864
8.178HxInstantiatorAdd Class Template Reference 865
8.179HxInstantiatorAddReduce Class Template Reference 866
8.180HxInstantiatorAddSat Struct Template Reference 866
8.181HxInstantiatorAddV Class Template Reference 867
8.182HxInstantiatorAnd Class Template Reference 868
8.183HxInstantiatorAndV Class Template Reference 868
8.184HxInstantiatorArg Class Template Reference 869
8.185HxInstantiatorAsin Class Template Reference 869
8.186HxInstantiatorAtan Class Template Reference 870
8.187HxInstantiatorAtan2 Class Template Reference 871
8.188HxInstantiatorCeil Class Template Reference 871
8.189HxInstantiatorColSpace Class Template Reference 872
8.190HxInstantiatorComplement Class Template Reference 872
8.191HxInstantiatorConjugate Class Template Reference 873
8.192HxInstantiatorCos Class Template Reference 874
8.193HxInstantiatorCosh Class Template Reference 874
8.194HxInstantiatorCross Class Template Reference 875
8.195HxInstantiatorCrossV Class Template Reference 875
8.196HxInstantiatorDiv Class Template Reference 876
8.197HxInstantiatorDivV Class Template Reference 877
8.198HxInstantiatorDot Class Template Reference 877
8.199HxInstantiatorDotV Class Template Reference 878
8.200HxInstantiatorEqual Class Template Reference 879
8.201HxInstantiatorEqualV Class Template Reference 879
8.202HxInstantiatorExp Class Template Reference 880
8.203HxInstantiatorExpPix Class Template Reference 880
8.204HxInstantiatorFloor Class Template Reference 881
8.205HxInstantiatorGpi Class Template Reference 882
8.206HxInstantiatorGreaterEqual Class Template Reference 882
8.207HxInstantiatorGreaterEqualV Class Template Reference 883
8.208HxInstantiatorGreaterThan Class Template Reference 883

8.209HxInstantiatorGreaterThanV Class Template Reference 884
8.210HxInstantiatorHighlightRegion Class Template Reference 885
8.211HxInstantiatorImpPix Class Template Reference 885
8.212HxInstantiatorInf Class Template Reference 886
8.213HxInstantiatorInfReduce Class Template Reference 887
8.214HxInstantiatorInfV Class Template Reference 887
8.215HxInstantiatorLeftShift Class Template Reference 888
8.216HxInstantiatorLeftShiftV Class Template Reference 889
8.217HxInstantiatorLessEqual Class Template Reference 889
8.218HxInstantiatorLessEqualV Class Template Reference 890
8.219HxInstantiatorLessThan Class Template Reference 890
8.220HxInstantiatorLessThanV Class Template Reference 891
8.221HxInstantiatorLog Class Template Reference 892
8.222HxInstantiatorLog10 Class Template Reference 892
8.223HxInstantiatorMax Class Template Reference 893
8.224HxInstantiatorMaxReduce Class Template Reference 894
8.225HxInstantiatorMaxV Class Template Reference 894
8.226HxInstantiatorMin Class Template Reference 895
8.227HxInstantiatorMinReduce Class Template Reference 895
8.228HxInstantiatorMinV Class Template Reference 896
8.229HxInstantiatorMod Class Template Reference 897
8.230HxInstantiatorModV Class Template Reference 897
8.231HxInstantiatorMul Class Template Reference 898
8.232HxInstantiatorMulReduce Class Template Reference 899
8.233HxInstantiatorMulV Class Template Reference 899
8.234HxInstantiatorNegate Class Template Reference 900
8.235HxInstantiatorNorm1 Class Template Reference 901
8.236HxInstantiatorNorm2 Class Template Reference 901
8.237HxInstantiatorNorm2Sqr Class Template Reference 902
8.238HxInstantiatorNormInf Class Template Reference 903
8.239HxInstantiatorNotEqual Class Template Reference 903
8.240HxInstantiatorNotEqualV Class Template Reference 904
8.241HxInstantiatorOr Class Template Reference 904
8.242HxInstantiatorOrV Class Template Reference 905
8.243HxInstantiatorPow Class Template Reference 906
8.244HxInstantiatorPowV Class Template Reference 906

8.245HxInstantiatorProduct Class Template Reference 907
8.246HxInstantiatorRGB2Intensity Class Template Reference 908
8.247HxInstantiatorRightShift Class Template Reference 908
8.248HxInstantiatorRightShiftV Class Template Reference 909
8.249HxInstantiatorRound Class Template Reference 910
8.250HxInstantiatorSet Struct Template Reference 910
8.251HxInstantiatorSetPartImg Struct Template Reference 914
8.252HxInstantiatorSetVal Class Template Reference 915
8.253HxInstantiatorSin Class Template Reference 915
8.254HxInstantiatorSinh Class Template Reference 916
8.255HxInstantiatorSpi Class Template Reference 916
8.256HxInstantiatorSqrDst Struct Template Reference 917
8.257HxInstantiatorSqrt Class Template Reference 918
8.258HxInstantiatorSub Class Template Reference 918
8.259HxInstantiatorSubSat Struct Template Reference 919
8.260HxInstantiatorSubV Class Template Reference 920
8.261HxInstantiatorSum Class Template Reference 920
8.262HxInstantiatorSup Class Template Reference 921
8.263HxInstantiatorSupReduce Class Template Reference 921
8.264HxInstantiatorSupV Class Template Reference 922
8.265HxInstantiatorTan Class Template Reference 923
8.266HxInstantiatorTanh Class Template Reference 923
8.267HxInstantiatorTriStateThreshold Struct Template Reference 924
8.268HxInstantiatorUpoMax Class Template Reference 924
8.269HxInstantiatorUpoMin Class Template Reference 925
8.270HxInstantiatorUpoThreshold Class Template Reference 926
8.271HxInstantiatorVec2 Class Template Reference 926
8.272HxInstantiatorVec3 Class Template Reference 927
8.273HxInstantiatorXor Class Template Reference 927
8.274HxInstantiatorXorV Class Template Reference 928
8.275HxInstDiyTranspose Class Template Reference 929
8.276HxInstExportExtraIdentMaskCentralMoments Struct Template Reference 929
8.277HxInstExportExtraIdentMaskMean Struct Template Reference 930
8.278HxInstExportExtraIdentMaskMedian Struct Template Reference 931
8.279HxInstExportExtraIdentMaskMoments Struct Template Reference 931
8.280HxInstExportExtraIdentMaskStdev Struct Template Reference 932

8.281HxInstExportExtraIdentMaskSum Struct Template Reference 933
8.282HxInstExportExtraWeightMaskSum Struct Template Reference 934
8.283HxInstExpPpm Class Template Reference 934
8.284HxInstGenConv2dAddInf Class Template Reference 935
8.285HxInstGenConv2dAddMax Class Template Reference 936
8.286HxInstGenConv2dAddMin Class Template Reference 936
8.287HxInstGenConv2dAddSup Class Template Reference 937
8.288HxInstGenConv2dK1dAddInf Class Template Reference 938
8.289HxInstGenConv2dK1dAddMax Class Template Reference 939
8.290HxInstGenConv2dK1dAddMin Class Template Reference 939
8.291HxInstGenConv2dK1dAddSup Class Template Reference 940
8.292HxInstGenConv2dK1dMulAdd Class Template Reference 941
8.293HxInstGenConv2dMulAdd Class Template Reference 942
8.294HxInstGenConv2dSepAddInf Class Template Reference 942
8.295HxInstGenConv2dSepAddMax Class Template Reference 943
8.296HxInstGenConv2dSepAddMin Class Template Reference 944
8.297HxInstGenConv2dSepAddSup Class Template Reference 945
8.298HxInstGenConv2dSepMulAdd Class Template Reference 945
8.299HxInstGenConv3dK1dMulAdd Class Template Reference 946
8.300HxInstGenConv3dMulAdd Class Template Reference 947
8.301HxInstGeneratePix Class Template Reference 948
8.302HxInstImpBytes Class Template Reference 948
8.303HxInstImpPackRgb Class Template Reference 949
8.304HxInstImpPpm Class Template Reference 949
8.305HxInstInOutGetPoints Class Template Reference 950
8.306HxInstKerNgb2dNormCorrelation Class Template Reference 950
8.307HxInstNgb2dMean Class Template Reference 951
8.308HxInstNgbIsMaxGradDir2d Class Template Reference 952
8.309HxInstNgbLWshed2d Struct Template Reference 952
8.310HxInstNgbNonMaxSuppression2d Class Template Reference 953
8.311HxInstNgbPercentile2d Class Template Reference 954
8.312HxInstRecGenConv2dAddMin Class Template Reference 954
8.313HxInstRecGenConv2dK1dAddMin Class Template Reference 955
8.314HxInstRecGenConv2dK1dMulAdd Class Template Reference 956
8.315HxInstRecGenConv2dMulAdd Class Template Reference 956
8.316HxInstRgb2dBinary Class Template Reference 957

8.317HxInstRgb2dCMY Class Template Reference 958
8.318HxInstRgb2dDirect Class Template Reference 958
8.319HxInstRgb2dDirectNC Class Template Reference 959
8.320HxInstRgb2dHSI Class Template Reference 959
8.321HxInstRgb2dLab Class Template Reference 960
8.322HxInstRgb2dLabel Class Template Reference 961
8.323HxInstRgb2dLogMag Class Template Reference 961
8.324HxInstRgb2dLuv Class Template Reference 962
8.325HxInstRgb2dOOO Class Template Reference 962
8.326HxInstRgb2dStretch Class Template Reference 963
8.327HxInstRgb2dXYZ Class Template Reference 964
8.328HxInstRgb3dBinary Class Template Reference 964
8.329HxInstRgb3dCMY Class Template Reference 965
8.330HxInstRgb3dDirect Class Template Reference 965
8.331HxInstRgb3dHSI Class Template Reference 966
8.332HxInstRgb3dLab Class Template Reference 967
8.333HxInstRgb3dLabel Class Template Reference 967
8.334HxInstRgb3dLogMag Class Template Reference 968
8.335HxInstRgb3dLuv Class Template Reference 968
8.336HxInstRgb3dOOO Class Template Reference 969
8.337HxInstRgb3dStretch Class Template Reference 970
8.338HxInstRgb3dXYZ Class Template Reference 970
8.339HxKernel1d Class Template Reference 971
8.340HxKernel2d Class Template Reference 971
8.341HxKernel3d Class Template Reference 972
8.342HxKerNgbNormCorrelation Class Template Reference 973
8.343HxLocalInterpol Class Reference 976
8.344HxMatrix Class Reference 979
8.345HxMfBpo Class Reference 1006
8.346HxMfDiy Class Reference 1009
8.347HxMfExportExtra Class Reference 1011
8.348HxMfGenConv Class Reference 1013
8.349HxMfIdentity Class Reference 1016
8.350HxMfKernelNgb Class Reference 1018
8.351HxMfMNpo Class Reference 1020
8.352HxMfMpo Class Reference 1024

8.353HxMfNgb Class Reference 1026
8.354HxMfQueueBased Class Reference 1029
8.355HxMfResize Class Reference 1031
8.356HxMfUpo Class Reference 1033
8.357HxNameTable Class Reference 1035
8.358HxNgbBernsen Class Template Reference 1038
8.359HxNgbDefuz Class Template Reference 1040
8.360HxNgbHilditch Class Template Reference 1042
8.361HxNgbIsMaxGradDir2d Class Template Reference 1044
8.362HxNgbKuwahara Class Template Reference 1048
8.363HxNgbLocalMode Class Template Reference 1049
8.364HxNgbLocalModeInst Class Template Reference 1052
8.365HxNgbLWshed2d Class Template Reference 1053
8.366HxNgbNonMaxSuppression2d Class Template Reference 1056
8.367HxNgbOpticalFlowInst Class Template Reference 1060
8.368HxNgbPercentile2d Class Template Reference 1060
8.369HxNJet Class Reference 1063
8.370HxPixelAllocator Class Template Reference 1072
8.371HxPointAndValue Class Reference 1073
8.372HxPointList Class Reference 1073
8.373HxPointR2 Class Reference 1074
8.374HxPointZ Class Reference 1077
8.375HxPointZList Class Reference 1078
8.376HxPolyline2d Class Reference 1079
8.377HxRcObject Class Reference 1082
8.378HxRcPtr Class Template Reference 1083
8.379HxRegData Class Reference 1084
8.380HxRegistry Class Reference 1088
8.381HxRegistryImporter Class Reference 1095
8.382HxRegKey Class Reference 1097
8.383HxRegKeyList Class Reference 1104
8.384HxRegValue Class Reference 1104
8.385HxRegValueList Class Reference 1107
8.386HxRgbBinary Class Template Reference 1107
8.387HxRgbCMY Class Template Reference 1109
8.388HxRgbDirect Class Template Reference 1111

8.389HxRgbDirectNC Class Template Reference 1112
8.390HxRgbHSI Class Template Reference 1114
8.391HxRgbLab Class Template Reference 1116
8.392HxRgbLabel Class Template Reference 1118
8.393HxRgbLogMag Class Template Reference 1120
8.394HxRgbLuv Class Template Reference 1122
8.395HxRgbOOO Class Template Reference 1123
8.396HxRgbStretch Class Template Reference 1125
8.397HxRgbXYZ Class Template Reference 1127
8.398HxSampledBSplineCurve Class Reference 1129
8.399HxSampledBSplineInterval Class Reference 1142
8.400HxScalarDouble Class Reference 1145
8.401HxScalarInt Class Reference 1164
8.402HxSegmentation2d Class Reference 1182
8.403HxSF Class Reference 1186
8.404HxSFFactory Class Reference 1189
8.405HxStringList Class Reference 1191
8.406HxTag Class Reference 1192
8.407HxTag1Phase Struct Reference 1194
8.408HxTag2Phase Struct Reference 1195
8.409HxTagCnum Struct Reference 1195
8.410HxTagList Class Reference 1196
8.411HxTagLoop Struct Reference 1199
8.412HxTagNPhase Struct Reference 1200
8.413HxTagPixOpIn Struct Reference 1201
8.414HxTagPixOpOut Struct Reference 1201
8.415HxTagTem Class Template Reference 1202
8.416HxTagTransInVar Struct Reference 1204
8.417HxTagTransVar Struct Reference 1204
8.418HxUpoAbs Class Template Reference 1205
8.419HxUpoAcos Class Template Reference 1206
8.420HxUpoArg Class Template Reference 1208
8.421HxUpoAsin Class Template Reference 1209
8.422HxUpoAtan Class Template Reference 1211
8.423HxUpoAtan2 Class Template Reference 1212
8.424HxUpoCeil Class Template Reference 1214

8.425HxUpoColSpace Class Template Reference 1215
8.426HxUpoComplement Class Template Reference 1217
8.427HxUpoConjugate Class Template Reference 1218
8.428HxUpoCos Class Template Reference 1220
8.429HxUpoCosh Class Template Reference 1221
8.430HxUpoExp Class Template Reference 1223
8.431HxUpoFloor Class Template Reference 1224
8.432HxUpoLog Class Template Reference 1226
8.433HxUpoLog10 Class Template Reference 1227
8.434HxUpoMax Class Template Reference 1229
8.435HxUpoMin Class Template Reference 1230
8.436HxUpoNegate Class Template Reference 1232
8.437HxUpoNorm1 Class Template Reference 1233
8.438HxUpoNorm2 Class Template Reference 1235
8.439HxUpoNorm2Sqr Class Template Reference 1236
8.440HxUpoNormInf Class Template Reference 1238
8.441HxUpoProduct Class Template Reference 1239
8.442HxUpoRound Class Template Reference 1241
8.443HxUpoSin Class Template Reference 1242
8.444HxUpoSinh Class Template Reference 1244
8.445HxUpoSqrt Class Template Reference 1245
8.446HxUpoSum Class Template Reference 1247
8.447HxUpoTan Class Template Reference 1248
8.448HxUpoTanh Class Template Reference 1250
8.449HxUpoTriStateThreshold Class Template Reference 1251
8.450HxValue Class Reference 1253
8.451HxValueList Class Reference 1261
8.452HxVec2Double Class Reference 1262
8.453HxVec2Int Class Reference 1281
8.454HxVec2Tem Class Template Reference 1300
8.455HxVec3Double Class Reference 1301
8.456HxVec3Int Class Reference 1321
8.457HxVec3Tem Class Template Reference 1341
8.458HxVector Class Reference 1342
8.459HxVectorR2 Class Reference 1356
8.460QThinning Class Template Reference 1360

8.461RGB2Intensity Class Template Reference 1361
8.462VideoReader Class Reference 1363
8.463VxStructureEval Struct Reference 1363

Chapter 1

Class Overview

1.1 Class overview

The most important user level classes:

HxImageRep (p. 620), **HxImageFactory** (p. 605), **HxImageSeq** (p. 646), **HxHistogram** (p. 548), **HxNJet** (p. 1063), **HxBSplineCurve** (p. 477), **HxSampledBSplineCurve** (p. 1129)

Basic classes and types:

HxScalarInt (p. 1164), **HxScalarDouble** (p. 1145), **HxVec2Int** (p. 1281), **HxVec2Double** (p. 1262), **HxVec3Int** (p. 1321), and **HxVec3Double** (p. 1301).

HxPoint (p. 325), **HxPointInt** (p. 325), **HxPointZ** (p. 1077), **HxPointList.h**

HxString (p. 358), **HxStringList.h**

HxMatrix (p. 979), **HxVector** (p. 1342)

Image related classes:

HxImageSignature (p. 685), **HxGeoIntType** (p. 231), **HxSizes** (p. 352), **HxTagList** (p. 1196), **HxValue** (p. 1253), **HxValueList.h**, **HxValueType** (p. 381)

Chapter 2

Pixels

2.1 Pixels

Since pixels are the atomic elements in image processing we start with the little ones. Actually, this chapter is not really about pixels, but deals with the representation and manipulation of pixel values in Horus.

This chapter:

- **Pixel value representation** (p. 3)
- **Arithmetic data types** (p. 3)
- **Unary operations on pixel values** (p. 4)
- **Binary operations on pixel values** (p. 5)
- **Color operations on pixel values** (p. 6)

2.2 Pixel value representation

Horus supports the following set of pixel value representations :

- **scalar values**: **HxByte** (p. 93) (unsigned char), short, int, float, double
- **complex values**: **HxComplex** (p. 506)
- **vector of 2 scalars**: **HxVec2Byte** (p. 381), **HxVec2Short** (p. 382), **HxVec2Int** (p. 1281), **HxVec2Float** (p. 382), **HxVec2Double** (p. 1262).
- **vector of 3 scalars**: **HxVec3Byte** (p. 383), **HxVec3Short** (p. 384), **HxVec3Int** (p. 1321), **HxVec3Float** (p. 383), **HxVec3Double** (p. 1301).

The classes and native types for pixel representations have a common set of operations that is geared towards storage and retrieval of the pixel values from memory. However, you will seldom use the pixel representations directly as image processing algorithms in Horus are written as C++ template functors based on so-called arithmetic data types.

2.3 Arithmetic data types

The set of arithmetic data types is defined to handle all arithmetic manipulation of (pixel) values. The set of arithmetic data types is smaller than the set of pixel representations to avoid an explosion of instantiations

in the template code. The arithmetic data types do not reduce computational performance in that most current processors are not able to do arithmetic operations on the original pixel values anyway, e.g. they can not add or multiply bytes, shorts, and floats. And, even if the processor could do it, the compiler would probably not support it.

The current set of arithmetic data types includes seven elements:

HxScalarInt (p. 1164), **HxScalarDouble** (p. 1145), **HxComplex** (p. 506), **HxVec2Int** (p. 1281), **HxVec2Double** (p. 1262), **HxVec3Int** (p. 1321), and **HxVec3Double** (p. 1301).

Each element can be constructed in one of the following ways: (i) without parameters (value is undefined), (ii) from another value of the same type, or (iii) from a specific number of corresponding native data types (**HxScalarInt** (p. 1164): 1 int, **HxScalarDouble** (p. 1145): 1 double, **HxComplex** (p. 506): 2 double's, **HxVec2Int** (p. 1281): 2 int's, **HxVec2Double** (p. 1262): 2 double's, **HxVec3Int** (p. 1321): 3 int's, **HxVec3Double** (p. 1301): 3 double's). A data type can also be casted to each of the other six data types as shown in the table. No other construction/conversion paths will be defined to avoid ambiguities in expressions.

	ScalarInt i	ScalarDouble d	Complex $\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	Vec2Int $\begin{pmatrix} i_1 \\ i_2 \end{pmatrix}$	Vec2Double $\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	Vec3Int $\begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix}$	Vec3Double $\begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$
ScalarInt i		$d(i)$	$\begin{pmatrix} d(i) \\ 0 \end{pmatrix}$	$\begin{pmatrix} i \\ i \end{pmatrix}$	$\begin{pmatrix} d(i) \\ d(i) \end{pmatrix}$	$\begin{pmatrix} i \\ i \\ i \end{pmatrix}$	$\begin{pmatrix} d(i) \\ d(i) \\ d(i) \end{pmatrix}$
ScalarDouble d	$i(d)$		$\begin{pmatrix} d \\ 0 \end{pmatrix}$	$\begin{pmatrix} i(d) \\ i(d) \end{pmatrix}$	$\begin{pmatrix} d \\ d \end{pmatrix}$	$\begin{pmatrix} i(d) \\ i(d) \\ i(d) \end{pmatrix}$	$\begin{pmatrix} d \\ d \\ d \end{pmatrix}$
Complex $\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	$i(d_1)$	d_1		$\begin{pmatrix} i(d_1) \\ i(d_2) \end{pmatrix}$	$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	$\begin{pmatrix} i(d_1) \\ i(d_2) \\ 0 \end{pmatrix}$	$\begin{pmatrix} d_1 \\ d_2 \\ 0 \end{pmatrix}$
Vec2Int $\begin{pmatrix} i_1 \\ i_2 \end{pmatrix}$	i_1	$d(i_1)$	$\begin{pmatrix} d(i_1) \\ d(i_2) \end{pmatrix}$		$\begin{pmatrix} d(i_1) \\ d(i_2) \end{pmatrix}$	$\begin{pmatrix} i_1 \\ i_2 \\ 0 \end{pmatrix}$	$\begin{pmatrix} d(i_1) \\ d(i_2) \\ 0 \end{pmatrix}$
Vec2Double $\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	$i(d_1)$	d_1	$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	$\begin{pmatrix} i(d_1) \\ i(d_2) \end{pmatrix}$		$\begin{pmatrix} i(d_1) \\ i(d_2) \\ 0 \end{pmatrix}$	$\begin{pmatrix} d_1 \\ d_2 \\ 0 \end{pmatrix}$
Vec3Int $\begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix}$	i_1	$d(i_1)$	$\begin{pmatrix} d(i_1) \\ d(i_2) \end{pmatrix}$	$\begin{pmatrix} i_1 \\ i_2 \end{pmatrix}$	$\begin{pmatrix} d(i_1) \\ d(i_2) \end{pmatrix}$		$\begin{pmatrix} d(i_1) \\ d(i_2) \\ d(i_3) \end{pmatrix}$
Vec3Double $\begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$	$i(d_1)$	d_1	$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	$\begin{pmatrix} i(d_1) \\ i(d_2) \end{pmatrix}$	$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$	$\begin{pmatrix} i(d_1) \\ i(d_2) \\ i(d_3) \end{pmatrix}$	

The table shows conversions between arithmetic data types. An element in the table shows the result of casting the value in the leftmost column (on the same row) to the value on the top row (in the same column). *d(i)* is the normal promotion of `int` to `double` as defined in C++. *i(d)* is the normal conversion of `double` to `int`.

2.4 Unary operations on pixel values

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

The unary pixel operations are defined for all arithmetic datatypes.

2.5 Binary operations on pixel values

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

The binary pixel operations are defined for all arithmetic datatypes.

2.6 Color operations on pixel values

The supported color models are enumerated in **HxColorModel** (p. 107). Color semantics are defined in **HxColor** (p. 498).

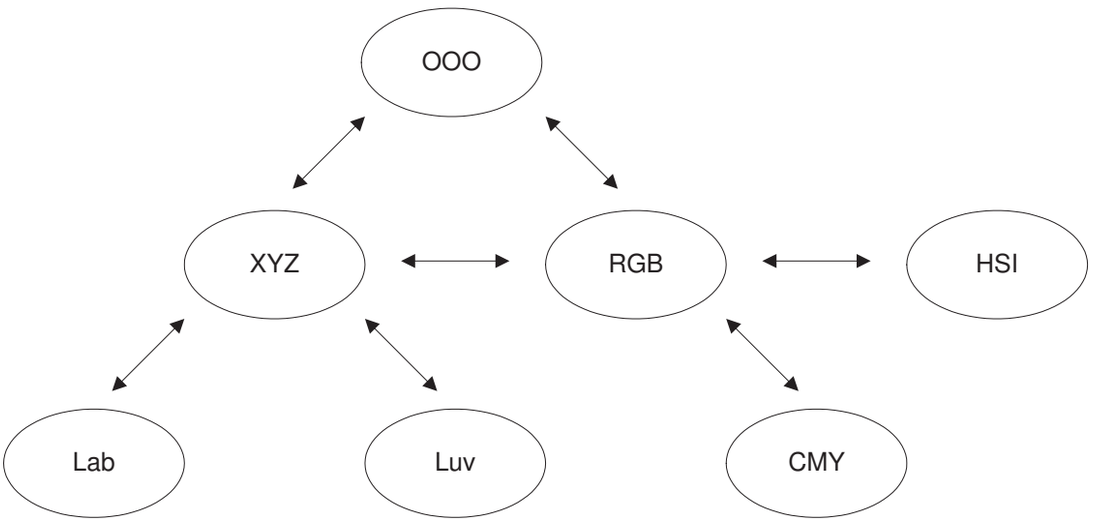


Figure 2.1: Color spaces

Functions for conversion of colors from one color space to another

RGB <-> CMY : **HxColRGB2CMY** (p. 101) and **HxColCMY2RGB** (p. 101).

RGB <-> XYZ : **HxColRGB2XYZ** (p. 101) and **HxColXYZ2RGB** (p. 102).

CMY <-> XYZ : **HxColCMY2XYZ** (p. 102) and **HxColXYZ2CMY** (p. 102).

Lab <-> XYZ : **HxColLab2XYZ** (p. 102) and **HxColXYZ2Lab** (p. 103).

Luv <-> XYZ : **HxColLuv2XYZ** (p. 103) and **HxColXYZ2Luv** (p. 103).

RGB <-> OOO : **HxColRGB2OOO** (p. 104) and **HxColOOO2RGB** (p. 104).

XYZ <-> OOO : **HxColXYZ2OOO** (p. 104) and **HxColOOO2XYZ** (p. 104).

RGB <-> HSI : **HxColRGB2HSI** (p. 105) and **HxColHSI2RGB** (p. 105).

Chapter 3

Images

3.1 Images

Our definition of images and operations on images is given in **Image definition and generic operations** (p. 8).

More down to earth, image data is stored in an **HxImageRep** (p. 620) and processed via functions such as listed in **Global image functions** (p. 10).

If you really want to know what is going on, see **Image data representation** (p. 12) and **The way image operations work** (p. 15).

This chapter:

- **Image definition and generic operations** (p. 8)
- **Global image functions** (p. 10)
- **Image data representation** (p. 12)
  - **Image signatures** (p. 13)
  - **Image data pointers** (p. 15)
- **The way image operations work** (p. 15)
  - **Image processing patterns and variations** (p. 24)
  - **Method frames** (p. 19)
  - **Border handling** (p. 20)
  - **Image functors** (p. 21)
  - **Image functor keys** (p. 23)
  - **Image functors in the registry** (p. 24)

3.2 Image definition and generic operations

3.3 Images and semantics

The use of images include:

- intensity images: the pixel value indicates a monochrome light intensity
- color images: the pixel value represents a color (RGB, HSI, etc.)

- X-ray, ultrasound, or electron microscope images: a pixel value depends on object density or another physical phenomena
- satellite images: the pixel value represents a recording of up to 7 spectral bands
- range images: the pixel value indicates a distance
- characteristic images: the pixel value indicates whether the pixel is element of a set
- flow fields: the pixel value represents a motion vector
- complex images: FFT domain

As stated in the introduction, we strive for a separation between semantics, representation, and implementation of objects. The representation is to provide a "neutral" view of a concept, a more or less mathematical view on the functionality needed in dealing with instances of the concept. From a software design point of view, representation and associated functionality is more important than semantics because it provides most opportunities for reusability. After all, you could say that semantics is nothing more than a label assigned to a representation.

Figure 3.1: Images in Horus

In Horus images are represented by the class **HxImageRep** (p. 620). The actual image data is stored in **HxImageData** (p. 581). The class `Image with semantics` is used to associate semantics with the pixel values, i.e. what does the numerical value of a pixel actually mean. The current implementation does not provide a class for semantics.

3.4 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.

3.5 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 ($N : \mathbf{X} \rightarrow 2^{\mathbf{X}}$) and Λ a reduce operation on the pixel values in a set ($\Lambda : (\mathbf{X}, \mathbf{F}) \rightarrow \mathbf{F}$). 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 $\mathbf{X}$.

3.6 Global image functions

Arithmetic unary **HxAbs** (p. 75), **HxCeil** (p. 97), **HxComplement** (p. 108), **HxExp** (p. 135), **HxFloor** (p. 144), **HxLog** (p. 273), **HxLog10** (p. 274), HxLut, **HxNegate** (p. 304), **HxNorm1** (p. 306), **HxNorm2** (p. 306), **HxNorm2Sqr** (p. 307), **HxNormInf** (p. 308), **HxProjectRange** (p. 329), **HxReciprocal** (p. 331), **HxSqrt** (p. 355), **HxRound** (p. 343), **HxUnaryMax** (p. 375), **HxUnaryMin** (p. 375), **HxUnaryProduct** (p. 376), **HxUnarySum** (p. 376)

trigonometric **HxAcos** (p. 76), **HxArg** (p. 89), **HxAsin** (p. 90), **HxAtan** (p. 91), **HxAtan2** (p. 92), **HxConjugate** (p. 111), **HxCos** (p. 119), **HxCosh** (p. 119), **HxSin** (p. 351), **HxSinh** (p. 352), **HxTan** (p. 367), **HxTanh** (p. 368).

binary **HxAdd** (p. 77), **HxAddSat** (p. 81), **HxAnd** (p. 86), **HxCross** (p. 120), **HxDiv** (p. 126), **HxDot** (p. 130), **HxEqual** (p. 133), **HxGreaterEqual** (p. 232), **HxGreaterThan** (p. 234), **HxInf** (p. 259), **HxInverseProjectRange** (p. 262), **HxLeftShift** (p. 265), **HxLessEqual** (p. 268), **HxLessThan** (p. 270), **HxMax** (p. 294), **HxMin** (p. 296), **HxMod** (p. 298), **HxMul** (p. 302), **HxNotEqual** (p. 309), **HxOr** (p. 315), **HxPow** (p. 327), **HxRightShift** (p. 340), **HxSub** (p. 359), **HxSubSat** (p. 360), **HxSup** (p. 362), **HxXor** (p. 396).

binary value **HxAddVal** (p. 82), HxAffinePix, **HxAndVal** (p. 87), **HxCrossVal** (p. 121), **HxDivVal** (p. 128), **HxDotVal** (p. 132), **HxEqualVal** (p. 134), **HxGreaterEqualVal** (p. 234), **HxGreaterThanVal** (p. 236), **HxInfVal** (p. 261), **HxLeftShiftVal** (p. 267), **HxLessEqualVal** (p. 269), **HxLessThanVal** (p. 271), **HxMaxVal** (p. 295), **HxMinVal** (p. 297), **HxModVal** (p. 299), **HxMulVal** (p. 303), **HxNotEqualVal** (p. 310), **HxOrVal** (p. 317), **HxPowVal** (p. 328), **HxRightShiftVal** (p. 341), **HxSubVal** (p. 361), **HxSupVal** (p. 364), **HxXorVal** (p. 397).

reduce **HxPixInf** (p. 321), **HxPixMax** (p. 321), **HxPixMin** (p. 322), **HxPixProduct** (p. 323), **HxPixSum** (p. 324), **HxPixSup** (p. 324).

Color **HxColorSpace** (p. 107), HxOpponentColor, **HxRGB2Intensity** (p. 339)

Conversion **HxImageAsByte** (p. 244), **HxImageAsComplex** (p. 245), **HxImageAsDouble** (p. 245), **HxImageAsFloat** (p. 246), **HxImageAsInt** (p. 247), **HxImageAsShort** (p. 248), **HxImageAsVec2Byte** (p. 248), **HxImageAsVec2Double** (p. 249), **HxImageAsVec2Float** (p. 250), **HxImageAsVec2Int** (p. 251), **HxImageAsVec2Short** (p. 251), **HxImageAsVec3Byte** (p. 252), **HxImageAsVec3Double** (p. 253), **HxImageAsVec3Float** (p. 254), **HxImageAsVec3Int** (p. 254), **HxImageAsVec3Short** (p. 255).

Detector HxImageToHistogram, HxImageToHistogramMask, HxGreyEdgeHistogram, HxLabelBlobs

Export **HxExportByteData** (p. 136), **HxExportDoubleData** (p. 136), **HxExportFloatData** (p. 137), **HxExportIntData** (p. 138), **HxExportMatlabPixels** (p. 139), HxExportPpmPixels, **HxExportShortData** (p. 140), **HxImagesToFile** (p. 258), HxMatrixConv, **HxWriteFile** (p. 395).

Filter **HxCannyEdgeMap** (p. 94), **HxCannyThreshold** (p. 95), **HxCannyThresholdAlt** (p. 96), **HxCannyThresholdRec** (p. 97), **HxConvGauss2d** (p. 113), **HxConvGauss3d** (p. 114), **HxConvKernelSeparated** (p. 116), **HxConvKernelSeparated2d** (p. 117), **HxConvolution** (p. 118), **HxDefuz** (p. 122), **HxDistanceTransform** (p. 125), **HxGauss** (p. 225), **HxGaussDerivative2d** (p. 226), **HxGaussDerivative3d** (p. 228), **HxGaussianDeblur** (p. 229), **HxLocalMode** (p. 272), **HxNonMaxSuppressionGradDir** (p. 305), **HxKuwahara** (p. 264), **HxNormalizedCorrelation** (p. 307), **HxPercentile** (p. 320), **HxRecGauss** (p. 330), **HxUniform** (p. 377), **HxUniformNonSep** (p. 378).

Generation **HxImagesFromFile** (p. 257), **HxMakeFrom2Images** (p. 275), **HxMakeFrom3Images** (p. 277), **HxMakeFromByteData** (p. 278), **HxMakeFromDoubleData** (p. 279), **HxMakeFromFile** (p. 280), **HxMakeFromFloatData** (p. 281), **HxMakeFromGrayValue** (p. 282), **HxMakeFromImage** (p. 283), **HxMakeFromImport** (p. 284), **HxMakeFromIntData** (p. 284), **HxMakeFromJavaRgb** (p. 285), **HxMakeFromMatlab** (p. 287), **HxMakeFromNamedGenerator** (p. 288), HxMakeFromPpmPixels, **HxMakeFromShortData** (p. 289), **HxMakeFromSignature** (p. 289), **HxMakeFromValue** (p. 290), **HxMakeGaussian1d** (p. 292), **HxMakeParabola1d** (p. 293)

Geometric **HxExtend** (p. 141), **HxExtendVal** (p. 142), **HxReflect** (p. 332), **HxRestrict** (p. 337), **HxRotate** (p. 342), **HxScale** (p. 344), **HxTranslate** (p. 371), **HxTranspose** (p. 373).

Inquiry **HxImageMaxSize** (p. 256), **HxImageMinSize** (p. 257).

Mask **HxIdentMaskCentralMoments** (p. 238), **HxIdentMaskMean** (p. 239), **HxIdentMaskMedian** (p. 240), **HxIdentMaskMoments** (p. 241), **HxIdentMaskStDev** (p. 242), **HxIdentMaskSum** (p. 242), **HxIdentMaskVariance** (p. 243), **HxWeightMaskSum** (p. 394).

Morphology HxAreaClosing, HxAreaOpening, HxClosing, **HxClosingByReconstruction** (p. 98), **HxClosingByReconstructionTopHat** (p. 99), **HxClosingTopHat** (p. 99), **HxConditionalDilation** (p. 109), **HxConditionalErosion** (p. 110), HxDilation, **HxDistanceTransformMM** (p. 125), HxErosion, **HxGeodesicDistanceTransform** (p. 230), **HxHilditchSkeleton** (p. 237), **HxHitOrMiss** (p. 238), **HxInfimumReconstruction** (p. 261), HxMorphologicalContour, **HxMorphologicalGradient** (p. 301), **HxMorphologicalGradient2** (p. 301), HxOpening, HxOpeningByReconstruction, **HxOpeningByReconstructionTopHat** (p. 311), **HxOpeningTopHat** (p. 312), **HxParabolicDilation** (p. 318), **HxParabolicErosion** (p. 319), HxPeakRemoval, **HxRegionalMaxima** (p. 333), **HxRegionalMinima** (p. 334), **HxSKIZ** (p. 354), **HxSkeleton** (p. 353), **HxSupremumReconstruction** (p. 363), **HxThickening** (p. 368), **HxThinning** (p. 369), HxValleyRemoval, **HxWatershed** (p. 384), **HxWatershedMarkers** (p. 385), **HxWatershedMarkers2** (p. 386), **HxWatershedSlow** (p. 387),

Motion **HxDisplayOF** (p. 123), HxOpticalFlow, **HxOpticalFlowMultiScale** (p. 313),

Noise **HxAddBinaryNoise** (p. 79), **HxAddGaussianNoise** (p. 79), **HxAddPoissonNoise** (p. 80), **HxAddUniformNoise** (p. 81)

Pixel **HxContrastStretch** (p. 112), **HxSetBorderValue** (p. 349), **HxSetPartImage** (p. 350), **HxSquaredDistance** (p. 356)

Sample HxGetPoints, HxGetValues,

Segmentation HxBernsenThreshold, HxEntropyThreshold, HxIsodataThreshold, **HxLabel** (p. 264), **HxLabel2** (p. 265), **HxThreshold** (p. 370), **HxTriStateThreshold** (p. 374).

3.7 Image data representation

This section will be updated when the move to image functors has been completed.

The **HxImageRep** (p. 620) and **HxImageData** (p. 581) classes, etc.

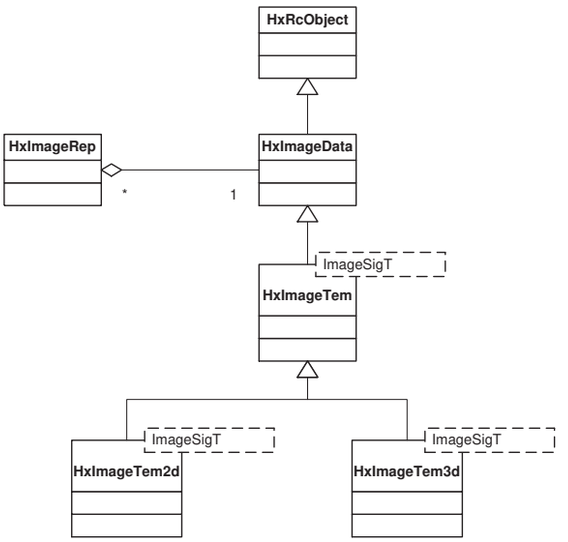


Figure 3.2: Image class hierarchy

HxImageFactory (p. 605), HxImageCreator, and HxDataPtrCreator.

HxPixelAllocator (p. 1072)

Wich image types are available : **Image signatures** (p. 13) and instantiations.

3.7.1 Image signatures

The signature of an image gives a description of the characteristics of an image type (a class). Part of the description contains runtime information and is universally accessible through the base class **HxImageSignature** (p. 685). The other part contains compile time information and is accessible only in specializations of **HxImageSignature** (p. 685).

The universally accessible part is obtained by members of **HxImageSignature** (p. 685):

- **HxImageSignature::imageDimensionality** (p. 691) : The dimensionality of an image, currently 1, 2, or 3.
- **HxImageSignature::pixelDimensionality** (p. 691) : The dimensionality of the pixel values in the image, currently 1 (scalar value), 2 (vector of 2 scalars), or 3 (vector of 3 scalars).
- **HxImageSignature::pixelType** (p. 691) : The type of the pixel values: INT, REAL, or COMPLEX.
- **HxImageSignature::pixelPrecision** (p. 691) : The number of bits used in the representation of a pixel value, currently 8, 16, 32, or 64.

Specializations of **HxImageSignature** (p. 685) provide additional information on the static types needed in the implementation of image processing operations by means of typedefs:

- **PixelType** The type of the actual pixel values stored in the image (see **Pixel value representation** (p. 3)).
- **ArithType** The type used in the evaluation of arithmetic operations on pixel values (see **Arithmetic data types** (p. 3)).

- **ArithTypeDouble** The type used in the evaluation of arithmetic operations on pixels in combination with other values that are represented as reals, e.g. in template operations with a kernel specified in `doubles`. May be the same as ArithType.
- **DataPtrType** The type of a data pointer to this image type (see **Image data pointers** (p. 15)).
- **Allocator** The type of a pixel allocator for this image type.
- **ProjectDomainImageSigType** The type (signature) of an image that is the result of the domain projection operation.
- **ArithImageSigType** The signature with the same image dimensionality but based on ArithType instead of PixelType.
- **ArithImageSigTypeDouble** The signature with the same image dimensionality but based on ArithTypeDouble.

The list of signatures

2D images with pixels represented by a scalar value **HxImageSig2dByte** (p. 669), **HxImageSig2dDouble** (p. 671), **HxImageSig2dFloat** (p. 672), **HxImageSig2dInt** (p. 672), **HxImageSig2dShort** (p. 673).

2D images with pixels represented by a complex number **HxImageSig2dComplex** (p. 670).

2D images with pixels represented by a vector of 2 scalars **HxImageSig2dVec2Byte** (p. 674), **HxImageSig2dVec2Double** (p. 675), **HxImageSig2dVec2Float** (p. 675), **HxImageSig2dVec2Int** (p. 676), **HxImageSig2dVec2Short** (p. 677).

2D images with pixels represented by a vector of 3 scalars **HxImageSig2dVec3Byte** (p. 678), **HxImageSig2dVec3Double** (p. 678), **HxImageSig2dVec3Float** (p. 679), **HxImageSig2dVec3Int** (p. 680), **HxImageSig2dVec3Short** (p. 681).

3D images with pixels represented by a scalar value **HxImageSig3dByte** (p. 681), **HxImageSig3dDouble** (p. 682), **HxImageSig3dFloat** (p. 683), **HxImageSig3dInt** (p. 684), **HxImageSig3dShort** (p. 684).

An overview of the currently available image types and the associated PixelType, ArithType, and ArithTypeDouble.

ImageType	PixelType	ArithType	ArithTypeDouble
HxImage2dByte	**HxByte** (p. 93)	**HxScalarInt** (p. 1164)	**HxScalarDouble** (p. 1145)
HxImage2dShort	short	**HxScalarInt** (p. 1164)	**HxScalarDouble** (p. 1145)
HxImage2dInt	int	**HxScalarInt** (p. 1164)	**HxScalarDouble** (p. 1145)
HxImage2dFloat	float	**HxScalarDouble** (p. 1145)	**HxScalarDouble** (p. 1145)
HxImage2dDouble	double	**HxScalarDouble** (p. 1145)	**HxScalarDouble** (p. 1145)
HxImage2dComplex	**HxComplex** (p. 506)	**HxComplex** (p. 506)	**HxComplex** (p. 506)
HxImage2dVec2Byte	**HxVec2Byte** (p. 381)	**HxVec2Int** (p. 1281)	**HxVec2Double** (p. 1262)
HxImage2dVec2Short	**HxVec2Short** (p. 382)	**HxVec2Int** (p. 1281)	**HxVec2Double** (p. 1262)
HxImage2dVec2Int	**HxVec2Int** (p. 1281)	**HxVec2Int** (p. 1281)	**HxVec2Double** (p. 1262)
HxImage2dVec2Float	**HxVec2Float** (p. 382)	**HxVec2Double** (p. 1262)	**HxVec2Double** (p. 1262)
HxImage2d-Vec2Double	**HxVec2Double** (p. 1262)	**HxVec2Double** (p. 1262)	**HxVec2Double** (p. 1262)
HxImage2dVec3Byte	**HxVec3Byte** (p. 383)	**HxVec3Int** (p. 1321)	**HxVec3Double** (p. 1301)
HxImage2dVec3Short	**HxVec3Short** (p. 384)	**HxVec3Int** (p. 1321)	**HxVec3Double** (p. 1301)
HxImage2dVec3Int	**HxVec3Int** (p. 1321)	**HxVec3Int** (p. 1321)	**HxVec3Double** (p. 1301)
HxImage2dVec3Float	**HxVec3Float** (p. 383)	**HxVec3Double** (p. 1301)	**HxVec3Double** (p. 1301)
HxImage2d-Vec3Double	**HxVec3Double** (p. 1301)	**HxVec3Double** (p. 1301)	**HxVec3Double** (p. 1301)
HxImage3dByte	**HxByte** (p. 93)	**HxScalarInt** (p. 1164)	**HxScalarDouble** (p. 1145)
HxImage3dShort	short	**HxScalarInt** (p. 1164)	**HxScalarDouble** (p. 1145)
HxImage3dInt	int	**HxScalarInt** (p. 1164)	**HxScalarDouble** (p. 1145)
HxImage3dFloat	float	**HxScalarDouble** (p. 1145)	**HxScalarDouble** (p. 1145)
HxImage3dDouble	double	**HxScalarDouble** (p. 1145)	**HxScalarDouble** (p. 1145)

3.7.2 Image data pointers

Image data is accessed through data pointers (a variable of type `DataPtrType`). The `DataPtrType` is defined in the signature of an image (see **Image signatures** (p. 13)).

Typically, the data pointer type is (a specialization of) a template class with (trivial) inline member functions. The inline functions are eliminated through compiler optimization so there is no loss of efficiency in pixel manipulation. To keep the functions trivial and efficient issues such as border checking should not be done within member functions of the data pointer class.

The requirements for the `DataPtrType` expressed as a class definition are:

```
                        DataPtrType(const DataPtrType& rhs);

DataPtrType&            operator=(const DataPtrType& rhs);

void                    incX();
void                    decX();
void                    incY();
void                    decY();
void                    incZ();
void                    decZ();
void                    incX(int off);
void                    decX(int off);
void                    incY(int off);
void                    decY(int off);

void                    incXYZ(int xOff, int yOff, int zOff = 0);
void                    decXYZ(int xOff, int yOff, int zOff = 0);

ArithT                  read();
void                    write(const ArithT& val);
ArithT                  readIncX();
void                    writeIncX(const ArithT& val);

PixelT*                 data();
```

Horus provides three template classes fullfilling the requirements: **HxDataPtr2dScalarTem** (p. 526), **HxDataPtr2dTem** (p. 527), and **HxDataPtr3dScalarTem** (p. 528). The first two are optimized for 2D images, the last is for 3D images. There are two versions for 2D images (one for scalar pixel values and one for vector values) because of a small technical problem : the scalar values images use native types (short, int, float, and double) and one cannot define a constructor for these types.

Storage and retrieval of individual pixel values is done through the `read` and `write` member functions of the data pointer class. Outside the data pointer class pixel values are represented in one of the arithmetic data types (see **Arithmetic data types** (p. 3)).

3.8 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. 10) via the source code incorporated in the documentation.

(1) Global image functions : (HxAbs (p. 75)**)** Operations on images are invoked via a global function (see **Global image functions** (p. 10) for an overview). Basically, a global function is a wrapper around an **HxImageRep** (p. 620) member function that may do some prior initialization and translation of parameters.

So, **HxAbs** (p. 75) calls **HxImageRep::unaryPixOp** (p. 629).

(2) HxImageRep member functions (HxImageRep::unaryPixOp (p. 629)**)** The member functions of **HxImageRep** (p. 620) match the generic operations on images and their variations listed in **Image processing patterns and variations** (p. 24). The most important task of the member function itself

is to instantiate a method frame (see **Method frames** (p. 19) for an overview) to enfore the 'value' paradigm and to allow for heterogenous image types. To start the actual operation, the call is forwarded to **HxImageData** (p. 581).

So, **HxImageRep::unaryPixOp** (p. 629) calls **HxImageData::unaryPixOp** (p. 593).

(3) HxImageData member functions (HxImageData::unaryPixOp (p. 593)**)** The member functions of **HxImageData** (p. 581) try to locate the approriate image functor (see **Image functors** (p. 21) 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. 861).

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. 21)). The functor found is given the task of invoking the actual functor (the "real" functor, see **Image functors** (p. 21)) via callIt.

So, **HxImageData::unaryPixOp** (p. 593) calls **HxImgFtorI2::callIt** (p. 743).

(4) "I-nary" level image functors (HxImgFtorI2::callIt (p. 743)**)** 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. 749).

(5-1) "cast" level image functors (HxImgFtorI2Cast::callIt (p. 749)**)** At the "cast" level, the polymorphic **HxImageData** (p. 581) 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. 749) calls **HxImgFtorI2Cast::doIt** (p. 750).

(5-2) "cast" level image functors (HxImgFtorI2Cast::doIt (p. 750)**)** 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. 861).

(6) "real" image functors (HxImgFtorUpo::doIt (p. 861)**)** 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. 859) 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 a template parameter. Based on these typedefs, the appropriate global template function is selected to do the actual work. The selection is done by a separate global template function called a dispatch function.

So, **HxImgFtorUpo::doIt** (p. 861) calls **HxFuncUpoDispatch** (p. 225)

(7-1) global dispatch functions (HxFuncUpoDispatch (p. 225)**)** 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.

The unary pixel operation has two variations : translation variant and translation invariant. The variation is defined by the TransVarianceCategory typedef in the UpoT template parameter.

So **HxFuncUpoDispatch** (p. 225) calls **HxFuncUpo**(DstDataPtrT,SrcDataPtrT,HxSizes,UpoT&,HxTagTransInVar) (p. 224) or **HxFuncUpo**(DstDataPtrT,SrcDataPtrT,HxSizes,UpoT&,HxTagTransVar) (p. 225).

(7-2) global image processing functions (HxFuncUpo (p. 225)**)** 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 the user defined functor(s) 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 **HxInstantiatorAbs_c** (p. **??**) for the instantiations of our HxAbs example.

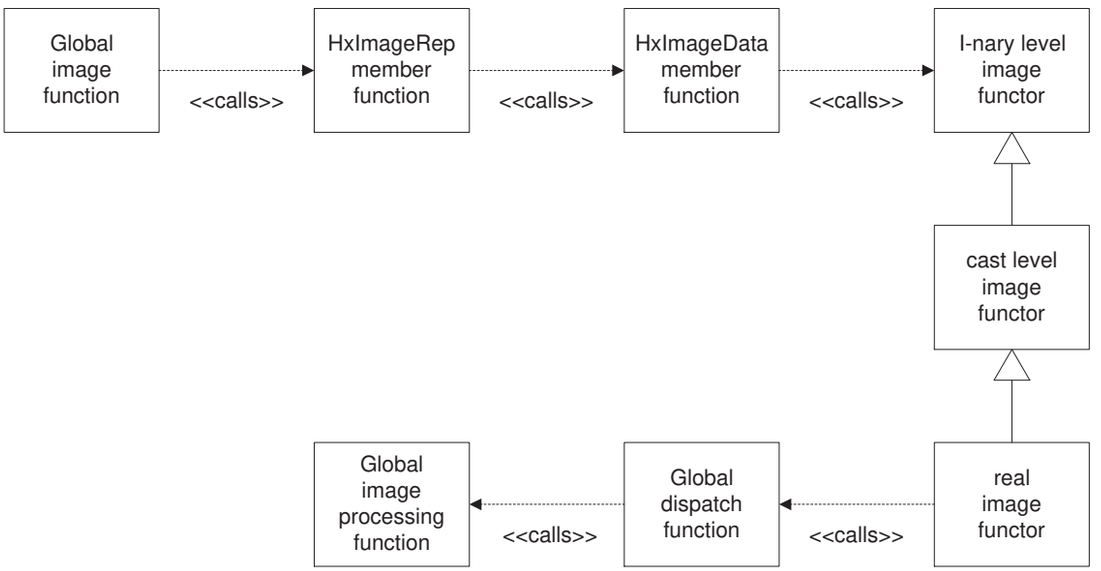


Figure 3.3: Graphical overview of the way image processing works

3.8.1 Image processing patterns and variations

The ”thing” that defines a pattern is the functor. Naming conventions for functors are geared towards the image parameters involved. This sometimes introduces confusion as, for example, a binary pixel operation applied to an image and a constant value is actually implemented by the unary pixel operation pattern (because there is only one image parameter involved).

- **Unary pixel operation** (p. 26)
  - **Translation invariant unary pixel operation** (p. 26)
  - **Translation variant unary pixel operation** (p. 27)
- **Binary pixel operation** (p. 28)
  - **Translation invariant binary pixel operation** (p. 28)
  - **Translation variant binary pixel operation** (p. 29)
- **Multi operation** (p. 29)
  - **Translation invariant multi pixel operation** (p. 30)
  - **Translation variant multi pixel operation** (p. 30)
- **M output, N input pixel operation** (p. 31)
  - **Translation invariant M output, N input pixel operation** (p. 31)
  - **Translation variant M output, N input pixel operation** (p. 32)
- **In/Out pixel operation** (p. 33)
  - **Translation invariant, 1 phase pixel export operation** (p. 33)
  - **Translation variant, 1 phase pixel export operation** (p. 34)

  - **Translation invariant, 1 phase pixel import operation** (p. 34)
  - **Translation variant, 1 phase pixel import operation** (p. 35)
  - **Translation invariant, n phase pixel export operation** (p. 35)
  - **Translation variant, n phase pixel export operation** (p. 36)
  - **Translation invariant, n phase pixel import operation** (p. 36)
  - **Translation variant, n phase pixel import operation** (p. 37)
- **Export operation with an extra image** (p. 38)
  - **Translation invariant, 1 phase export operation with an extra image** (p. 38)
  - **Translation variant, 1 phase export operation with an extra image** (p. 39)
  - **Translation invariant, N phase export operation with an extra image** (p. 39)
  - **Translation variant, N phase export operation with an extra image** (p. 40)
- Generalized convolution
  - **Generalized convolution on 2D images** (p. 41)
  - **Generalized convolution on 2D images, separated by dimension** (p. 41)
  - **Generalized convolution on 2D images, with a 1D kernel** (p. 42)
  - **Generalized convolution on 3D images** (p. 43)
  - **Generalized convolution on 3D images, with a 1D kernel** (p. 44)
- Recursive generalized convolution
  - **Recursive generalized convolution on 2D images** (p. 44)
  - **Recursive generalized convolution on 2D images, with a 1D kernel** (p. 45)
- **Neighbourhood operation on 2D images** (p. 45)
  - **1 phase, coordinate enumerated neighbourhood operation** (p. 46)
  - **1 phase, loop neighbourhood operation** (p. 47)
  - **2 phase, loop neighbourhood operation** (p. 48)
  - **N phase, loop neighbourhood operation** (p. 49)
- **Neighbourhood operation on 2D images with an extra image** (p. 49)
  - **1 phase, coordinate enumerated neighbourhood operation with an extra image** (p. 50)
  - **1 phase, loop neighbourhood operation with an extra image** (p. 51)
  - **2 phase, loop neighbourhood operation with an extra image** (p. 52)
  - **N phase, loop neighbourhood operation with an extra image** (p. 53)
- **Neighbourhood operation on 2D images with two extra images** (p. 53)
  - **1 phase, coordinate enumerated neighbourhood operation with two extra images** (p. 54)
  - **1 phase, loop neighbourhood operation with two extra images** (p. 55)
  - **2 phase, loop neighbourhood operation with two extra images** (p. 56)
  - **N phase, loop neighbourhood operation with two extra images** (p. 57)
- **Kernel based neighbourhood operation on 2D images** (p. 57)
  - **1 phase, loop neighbourhood operation with kernel** (p. 58)
  - **2 phase, loop neighbourhood operation using a kernel** (p. 59)
  - **N phase, loop neighbourhood operation using a kernel** (p. 60)
- **Do It Yourself operation** (p. 60)

Not addressable via the HxImageRep interface (for internal use only):

- **Set operation** (p. 61)
- Border set operation and variations
  - **Border set for 2D images** (p. 61)
  - **Border set for 3D images** (p. 62)

To be re-written:

- **Geometric operation on 2D images** (p. 62)
- Display RGB operation
  - **Display RGB operation for 2D images** (p. 62)
  - **Display RGB operation for 3D images** (p. 63)

3.8.2 Method frames

Method frames are used in **HxImageRep** (p. 620) member functions to enforce the 'value' or 'functional' paradigm. That is, each operation that would change the operand object results in a new object.

Method frames also faciliate operations on heterogeneous image types. In other words, method frames ensure that the image types (signatures) of all images involved in an operation "match". The image types involved in an operation are the object itself, all of its image parameters and the result of the operation. These types must match the signatures used to instantiate the image functor that is going to be called by the member function otherwise the operation cannot be executed. Method frames try to make a match based on information in the so-called rule base (**HxImgFtorRuleBase** (p. 840)). In case there is no exact match, the method frames will adjust image types by creating temporary variables of the appropriate image type and initializing them with data from the corresponding image. The method frames also allocate a result image of the appropriate type.

A method frame is initialized typically with a pointer to the data, i.e. an instantiation of **HxImageData** (p. 581), obtained via HxImageRep::pointee(). The constructor of the method frame takes care of possible allocation of temporary image data and conversion of image data according to the task of the frame. The specific copy and/or conversion schemes are hidden from the method frame class user by the `object`, `argument`, `kernel`, and `result` member functions. After construction of the method frame, the user should use only these functions to access the image data of the object as well as possible arguments. The result of the operation is typically an instantiation of **HxImageRep** (p. 620) intialized with the `result` of the method frame. The destructor of the method frame removes temporary images.

The list of method frames

- **HxMfBpo** (p. 1006)
- **HxMfDiy** (p. 1009)
- **HxMfExportExtra** (p. 1011)
- **HxMfGenConv** (p. 1013)
- **HxMfIdentity** (p. 1016)
- **HxMfKernelNgb** (p. 1018)
- **HxMfMNpo** (p. 1020)
- **HxMfMpo** (p. 1024)
- **HxMfNgb** (p. 1026)
- **HxMfResize** (p. 1031)
- **HxMfUpo** (p. 1033)

3.8.3 Border handling

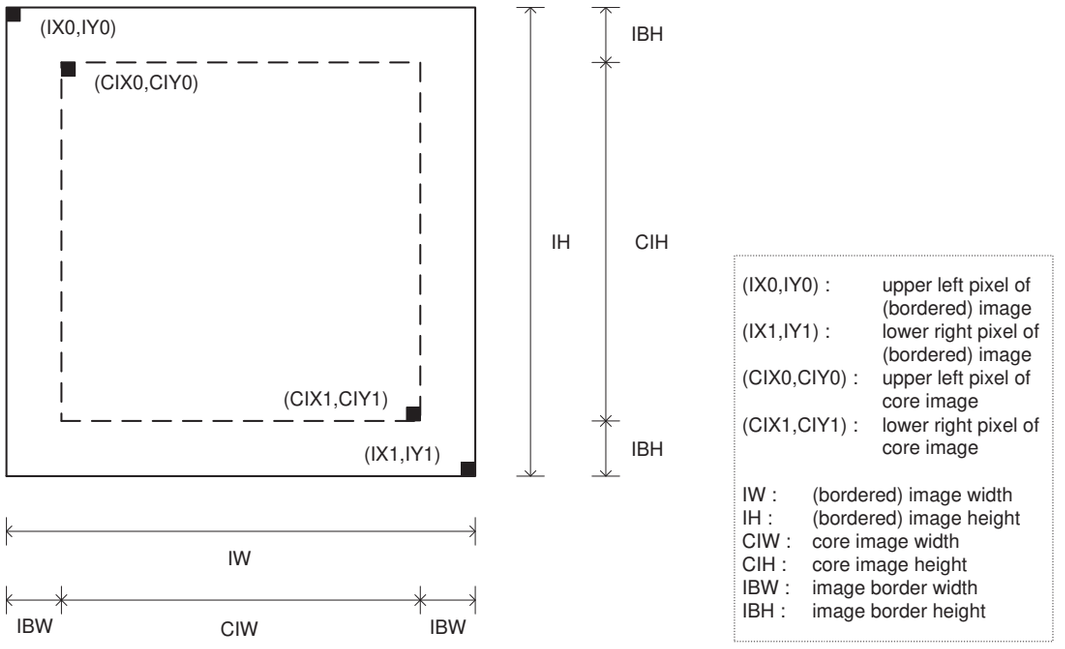


Figure 3.4: Image border definition

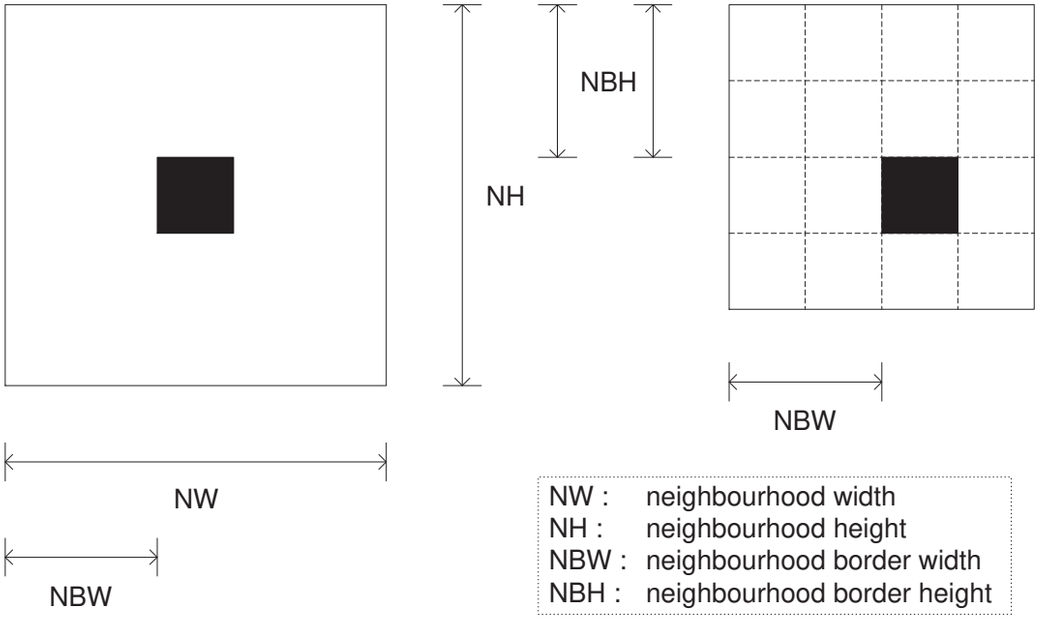


Figure 3.5: Neighbourhood border definition

- (bordered) image sizes : IS = (IW,IH)
- core image sizes : CIS = (CIW,CIH)
- image border sizes : IBS = (IBW,IBH)
- neighbourhood sizes : NS = (NW,NH)
- neighbourhood border sizes : NBS = (NBW,NBH)

3.8.4 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 (see **Image processing patterns and variations** (p. 24)) do what the user wants it to do.

Image functors are addressed via **Image functor keys** (p. 23). Keys are created in the constructor of the image functors at the "real functor" level. The various parts of the key are derived from the actual template parameters via **HxClassName** (p. 495).

Image functors are collected in the **HxImgFtorTable** (p. 856). **HxImgFtorTable** (p. 856) contains a map of <**HxImgFtorKey** (p. 799), **HxImgFunctor** (p. 862)*>. 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. 862) (the base class of the image functor hierarchy) inserts itself in the table under its key, i.e. it inserts the <key,object> combination in the table. The insert function puts a stringified version of the <key,functor> combination in the **HxRegistry** (p. 1088) under ["/imagefunctortable/entries"].

When the user calls an image processing fuction, the **HxImageData** (p. 581) member functions construct a key by converting their arguments to strings and use this key to retrieve the right image functor from the table. The **HxImageData** (p. 581) member functions use the template class **HxImgFtorTableTem** (p. 858) to get statically typed access at the level of "I-nary" functors (see below) in the image functor hierarchy. The downcast is "safe" because the key used to find the "real functor" is derived from the key corresponding to the "I-nary" functor. Naturally, this implies that the image functor and key hierarchies should be kept in close harmony.

Other classes:

HxImgFtorDescription (p. 704), **HxImgFtorRuleBase** (p. 840),

The image functor tree

The tree has four levels:

1. The root level : **HxImgFunctor** (p. 862) (for insertion and retrieval from the table).
2. The "I-nary" level : I1 stands for "taking 1 image parameter", I2 for 2 image parameters, etc.
3. The "cast" level : the highest template level for casting of polymorphic **HxImageData** (p. 581) parameters to statically typed image data pointers.
4. The "real functor" level : the leaf nodes are the actual functors that introduce additional, pattern specific template parameters.

- **HxImgFunctor** (p. 862)
  - **HxImgFtorI1** (p. 728)
    - **HxImgFtorI1Cast** (p. 731)<ImgSigT>
      - **HxImgFtorInOut** (p. 791)<ImgSigT, InOutT>
      - **HxImgFtorRgb2d** (p. 835)<ImgSigT, RgbT>
      - **HxImgFtorRgb3d** (p. 837)<ImgSigT, RgbT>

        - **HxImgFtorSetBorder2d** (p. 849)<ImgSigT>
        - **HxImgFtorSetBorder3d** (p. 852)<ImgSigT>
    - **HxImgFtorI2** (p. 737)
        - **HxImgFtorI2Cast** (p. 743)<DstImgSigT, SrcImgSigT>
            - **HxImgFtorExportExtra** (p. 708)<ImgSigT,ExtraImgSigT,ExportExtraT>
            - **HxImgFtorNgb2d** (p. 810)<DstImgSigT, SrcImgSigT, NgbT>
            - **HxImgFtorRecGenConv2d** (p. 828)<ImgSigT, KerImgSigT, PixOpT, RedOpT>
            - **HxImgFtorRecGenConv2dK1d** (p. 830)<ImgSigT, KerImgSigT, PixOpT, RedOpT>
            - **HxImgFtorSet** (p. 847)<DstImgSigT, SrcImgSigT>
            - HxImgFtorTranspose2d<DstImgSigT, SrcImgSigT>
            - **HxImgFtorUpo** (p. 859)<DstImgSigT, SrcImgSigT, UpoT>
    - **HxImgFtorI3** (p. 756)
        - **HxImgFtorI3Cast** (p. 759)<DstImgSigT, Src1ImgSigT, Src2ImgSigT>
            - **HxImgFtorBpo** (p. 700)<DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT>
            - **HxImgFtorGenConv2d** (p. 711)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
            - **HxImgFtorGenConv2dK1d** (p. 714)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
            - **HxImgFtorGenConv3d** (p. 721)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
            - **HxImgFtorGenConv3dK1d** (p. 723)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
            - **HxImgFtorKernelNgb2d** (p. 795)<DstImgSigT, SrcImgSigT, KerImgSigT, NgbT>
            - **HxImgFtorNgb2dExtra** (p. 813)<DstImgSigT, SrcImgSigT, ExtraImgSigT, NgbT>
    - **HxImgFtorI4** (p. 770)
        - **HxImgFtorI4Cast** (p. 772)<Img1SigT, Img2SigT, Img3SigT, Img4SigT>
            - **HxImgFtorGenConv2dSep** (p. 717)<DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT>
            - **HxImgFtorNgb2dExtra2** (p. 816)<DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT, NgbT>
    - **HxImgFtorIM** (p. 778)
        - **HxImgFtorIMCast** (p. 779)<DstImgSigT, SrcImgsSigT>
            - **HxImgFtorMpo** (p. 807)<DstImgSigT, SrcImgsSigT, MpoT>
    - **HxImgFtorIMN** (p. 784)
        - **HxImgFtorIMNCast** (p. 786)<DstImgsSigT, SrcImgsSigT>
            - **HxImgFtorMNpo** (p. 804)<DstImgSigT, SrcImgsSigT, MNpoT>

3.8.5 Image functor keys

A key consists of a list of strings corresponding to the name of an image functor and its template parameters. Concatenating the list of strings results in a string that resembles the ”real” image functor declaration, e.g. ”HxImgFtorBpo<Image2dInt8, Image2dInt32, Image2dInt32, greaterEqual>”

Construction of a key is initiated by the leaves in the tree. Strings know to the base classes are passed to the corresponding constructor. The remainder is added via **HxImgFtorKey::addArgument** (p. 801).

The image functor key tree

- **HxImgFtorKey** (p. 799)
    - **HxImgFtorI1Key** (p. 736)
        - **HxImgFtorI1CastKey** (p. 736)
            - **HxImgFtorInOutKey** (p. 794)
            - **HxImgFtorRgbKey** (p. 840)
            - **HxImgFtorSetBorderKey** (p. 854)

- **HxImgFtorI2Key** (p. 755)
  - **HxImgFtorI2CastKey** (p. 754)
    - **HxImgFtorExportExtra** (p. 708)
    - **HxImgFtorNgbKey** (p. 821)
    - **HxImgFtorRecGenConvKey** (p. 834)
    - **HxImgFtorRecGenConvK1dKey** (p. 833)
    - **HxImgFtorSetKey** (p. 855)
    - HxImgFtorTranspose2dKey
    - **HxImgFtorUpoKey** (p. 861)
- **HxImgFtorI3Key** (p. 768)
  - **HxImgFtorI3CastKey** (p. 767)
    - **HxImgFtorBpoKey** (p. 703)
    - **HxImgFtorGenConvKey** (p. 727)
    - **HxImgFtorGenConvK1dKey** (p. 726)
    - **HxImgFtorKernelNgbKey** (p. 798)
    - **HxImgFtorNgbExtraKey** (p. 820)
- **HxImgFtorI4Key** (p. 777)
  - **HxImgFtorI4CastKey** (p. 776)
    - **HxImgFtorGenConv2dSepKey** (p. 720)
    - **HxImgFtorNgbExtra2Key** (p. 819)
- **HxImgFtorIMKey** (p. 784)
  - **HxImgFtorIMCastKey** (p. 783)
    - **HxImgFtorMpoKey** (p. 809)
- **HxImgFtorIMNKey** (p. 790)
  - **HxImgFtorIMNCastKey** (p. 789)
    - **HxImgFtorMNpoKey** (p. 806)

Note that there is not a one-on-one mapping between keys and functors. Functors that differ only in the dimensionality of the algorithm typically use the same key.

3.8.6 Image functors in the registry

The Horus registry (see **Registry** (p. 72)) contains information on image functors.

Inserted and used by **HxImgFtorRuleBase** (p. 840):

```
[/]
[/imagefunctor]
[/imagefunctor/rulebase]
[/imagefunctor/rulebase/argumenttype]
[/imagefunctor/rulebase/extratype]
[/imagefunctor/rulebase/isModifying]
[/imagefunctor/rulebase/kerneltype]
[/imagefunctor/rulebase/resulttype]
```

Inserted by **HxImgFtorTable::insert** (p. 857):

```
[/imagefunctortable]
[/imagefunctortable/entries]
[/imagefunctortable/entries/HxImgFtorBpo]
[/imagefunctortable/entries/HxImgFtorExportExtra]
[/imagefunctortable/entries/HxImgFtorGenConv]
[/imagefunctortable/entries/HxImgFtorGenConvK1d]
[/imagefunctortable/entries/HxImgFtorInOut]
[/imagefunctortable/entries/HxImgFtorMpo]
[/imagefunctortable/entries/HxImgFtorNgb]
```

```
[/imagefunctortable/entries/HxImgFtorNgb2d]
[/imagefunctortable/entries/HxImgFtorRecNgb]
[/imagefunctortable/entries/HxImgFtorRgb]
[/imagefunctortable/entries/HxImgFtorSet]
[/imagefunctortable/entries/HxImgFtorSetBorder]
[/imagefunctortable/entries/HxImgFtorTranspose2d]
[/imagefunctortable/entries/HxImgFtorUpo]
```

Inserted by **HxImgFtorRgb2d::HxImgFtorRgb2d** (p. 836) and **HxImgFtorRgb3d::HxImgFtorRgb3d** (p. 838):

```
[/imagefunctortable/rgb]
```

3.8.7 Image processing patterns and variations

The ”thing” that defines a pattern is the functor. Naming conventions for functors are geared towards the image parameters involved. This sometimes introduces confusion as, for example, a binary pixel operation applied to an image and a constant value is actually implemented by the unary pixel operation pattern (because there is only one image parameter involved).

- **Unary pixel operation** (p. 26)
  - **Translation invariant unary pixel operation** (p. 26)
  - **Translation variant unary pixel operation** (p. 27)
- **Binary pixel operation** (p. 28)
  - **Translation invariant binary pixel operation** (p. 28)
  - **Translation variant binary pixel operation** (p. 29)
- **Multi operation** (p. 29)
  - **Translation invariant multi pixel operation** (p. 30)
  - **Translation variant multi pixel operation** (p. 30)
- **M output, N input pixel operation** (p. 31)
  - **Translation invariant M output, N input pixel operation** (p. 31)
  - **Translation variant M output, N input pixel operation** (p. 32)
- **In/Out pixel operation** (p. 33)
  - **Translation invariant, 1 phase pixel export operation** (p. 33)
  - **Translation variant, 1 phase pixel export operation** (p. 34)
  - **Translation invariant, 1 phase pixel import operation** (p. 34)
  - **Translation variant, 1 phase pixel import operation** (p. 35)
  - **Translation invariant, n phase pixel export operation** (p. 35)
  - **Translation variant, n phase pixel export operation** (p. 36)
  - **Translation invariant, n phase pixel import operation** (p. 36)
  - **Translation variant, n phase pixel import operation** (p. 37)
- **Export operation with an extra image** (p. 38)
  - **Translation invariant, 1 phase export operation with an extra image** (p. 38)
  - **Translation variant, 1 phase export operation with an extra image** (p. 39)
  - **Translation invariant, N phase export operation with an extra image** (p. 39)
  - **Translation variant, N phase export operation with an extra image** (p. 40)
- Generalized convolution
  - **Generalized convolution on 2D images** (p. 41)

  - **Generalized convolution on 2D images, separated by dimension** (p. 41)
  - **Generalized convolution on 2D images, with a 1D kernel** (p. 42)
  - **Generalized convolution on 3D images** (p. 43)
  - **Generalized convolution on 3D images, with a 1D kernel** (p. 44)
- Recursive generalized convolution
  - **Recursive generalized convolution on 2D images** (p. 44)
  - **Recursive generalized convolution on 2D images, with a 1D kernel** (p. 45)
- **Neighbourhood operation on 2D images** (p. 45)
  - **1 phase, coordinate enumerated neighbourhood operation** (p. 46)
  - **1 phase, loop neighbourhood operation** (p. 47)
  - **2 phase, loop neighbourhood operation** (p. 48)
  - **N phase, loop neighbourhood operation** (p. 49)
- **Neighbourhood operation on 2D images with an extra image** (p. 49)
  - **1 phase, coordinate enumerated neighbourhood operation with an extra image** (p. 50)
  - **1 phase, loop neighbourhood operation with an extra image** (p. 51)
  - **2 phase, loop neighbourhood operation with an extra image** (p. 52)
  - **N phase, loop neighbourhood operation with an extra image** (p. 53)
- **Neighbourhood operation on 2D images with two extra images** (p. 53)
  - **1 phase, coordinate enumerated neighbourhood operation with two extra images** (p. 54)
  - **1 phase, loop neighbourhood operation with two extra images** (p. 55)
  - **2 phase, loop neighbourhood operation with two extra images** (p. 56)
  - **N phase, loop neighbourhood operation with two extra images** (p. 57)
- **Kernel based neighbourhood operation on 2D images** (p. 57)
  - **1 phase, loop neighbourhood operation with kernel** (p. 58)
  - **2 phase, loop neighbourhood operation using a kernel** (p. 59)
  - **N phase, loop neighbourhood operation using a kernel** (p. 60)
- **Do It Yourself operation** (p. 60)

Not addressable via the HxImageRep interface (for internal use only):

- **Set operation** (p. 61)
- Border set operation and variations
  - **Border set for 2D images** (p. 61)
  - **Border set for 3D images** (p. 62)

To be re-written:

- **Geometric operation on 2D images** (p. 62)
- Display RGB operation
  - **Display RGB operation for 2D images** (p. 62)
  - **Display RGB operation for 3D images** (p. 63)

3.8.8 Unary pixel operation

The pattern loops over all pixels in the image and applies the pixel functor to each pixel. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::unaryPixOp** (p. 629)
- **HxImageRep::binaryPixOp**(const HxValue,HxString,HxTagList&) (p. 629)

Instantiation is done via the **HxImgFtorUpo** (p. 859) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- UpoT is the type of the pixel functor

The requirements for the UpoT parameter and an example of its use are given in the sections on variations:

- **Translation invariant unary pixel operation** (p. 26)
- **Translation variant unary pixel operation** (p. 27)

Tags for UpoT variations:

- TransVarianceCategory : **HxTagTransVar** (p. 1204) or **HxTagTransInVar** (p. 1204)

For further reference :

- Method frame : **HxMfUpo** (p. 1033)
- HxImageData member : **HxImageData::unaryPixOp** (p. 593)

3.8.9 Translation invariant unary pixel operation

Pseudo code of the operation:

```
UnaryPixOp(Dst, Src, Upo)
{
    foreach d, s in Dst, Src
        Dst(d) = Upo.doIt(Src(s));
}
```

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

```
template<class DstValT, class SrcValT>
class UpoT
{
public:
    typedef HxTagTransInVar     TransVarianceCategory;

                        UpoT(HxTagList&);

    DstValT             doIt(const SrcValT& v);

    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.

Example(s):

- instantiation : **HxInstantiatorAbs_c** (p. **??**), usage : **HxAbs** (p. 75)
- instantiation : **HxInstantiatorAddV_c** (p. **??**), usage : **HxAddVal** (p. 82)

3.8.10 Translation variant unary pixel operation

Pseudo code of the operation:

```
UnaryPixOp(Dst, Src, Upo)
{
    foreach d, s in Dst, Src
        Dst(d) = Upo.doIt(Src(s), s.x, s.y, s.z);
}
```

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

```
template<class DstValT, class SrcValT>
class UpoT
{
public:
    typedef HxTagTransVar       TransVarianceCategory;

                        UpoT(HxTagList&);

    DstValT             doIt(const SrcValT& v, int x, int y, int z);

    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.

3.8.11 Binary pixel operation

The pattern loops over all pixels in the input images and applies the pixel functor to each pair of pixels. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::binaryPixOp**(const HxImageRep,HxString,HxTagList&) (p. 630)

Instantiation is done via the **HxImgFtorBpo** (p. 700) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- Src1ImgSigT is the signature type of the first source image
- Src2ImgSigT is the signature type of the second source image
- BpoT is the type of the pixel functor

The requirements for the BpoT parameter and an example of its use are given in the sections on variations:

- **Translation invariant binary pixel operation** (p. 28)

- **Translation variant binary pixel operation** (p. 29)

Tags for BpoT variations:

- TransVarianceCategory : **HxTagTransVar** (p. 1204) or **HxTagTransInVar** (p. 1204)

For further reference :

- Method frame : **HxMfBpo** (p. 1006)
- HxImageData member : **HxImageData::binaryPixOp** (p. 593)

3.8.12 Translation invariant binary pixel operation

Pseudo code of the operation:

```
BinaryPixOp(Dst, Src1, Src2, Bpo)
{
    foreach d, s1, s2 in Dst, Src1, Src2
        Dst(d) = Bpo.doIt(Src1(s1), Src2(s2));
}
```

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

```
template<class DstValT, class Src1ValT, class Src2ValT>
class BpoT
{
public:
    typedef HxTagTransInVar     TransVarianceCategory;

                        BpoT(HxTagList&);

    DstValT             doIt(const Src1ValT& v1, const Src2ValT& v2);

    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.

Example(s):

- instantiation : **HxInstantiatorAdd_c** (p. **??**), usage : **HxAdd** (p. 77)

3.8.13 Translation variant binary pixel operation

Pseudo code of the operation:

```
BinaryPixOp(Dst, Src1, Src2, Bpo)
{
    foreach d, s1, s2 in Dst, Src1, Src2
        Dst(d) = Bpo.doIt(Src1(s1), Src2(s2), s1.x, s1.y, s1.z);
}
```

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

```
template<class DstValT, class Src1ValT, class Src2ValT>
class BpoT
{
public:
    typedef HxTagTransVar       TransVarianceCategory;

                        BpoT(HxTagList&);

    DstValT             doIt(const Src1ValT& v1, const Src2ValT& v2,
                             int x, int y, int z);

    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.

3.8.14 Multi operation

The pattern loops over all pixels in the input images and applies the pixel functor to each set of pixels. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::multiPixOp** (p. 630)

Instantiation is done via the **HxImgFtorMpo** (p. 807) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgsSigT is the signature type of the source images
- MpoT is the type of the multi pixel functor

The requirements for the MpoT parameter and an example of its use are given in the sections on variations:

- **Translation invariant multi pixel operation** (p. 30)
- **Translation variant multi pixel operation** (p. 30)

Tags for MpoT variations:

- TransVarianceCategory : **HxTagTransVar** (p. 1204) or **HxTagTransInVar** (p. 1204)

For further reference :

- Method frame : **HxMfMpo** (p. 1024)
- HxImageData member : **HxImageData::multiPixOp** (p. 594)

3.8.15 Translation invariant multi pixel operation

Pseudo code of the operation:

```
MultiPixOp(Dst, Src[], Mpo)
{
    foreach d, s, in Dst, Src[]
        Dst(d) = Mpo.doIt(Src[0](s) ... Src[N](s));
}
```

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

```
template<class DstValT, class SrcValT>
class MpoT
{
public:
    typedef HxTagTransInVar     TransVarianceCategory;

                    MpoT(HxTagList& tags);

    DstValT         doIt(SrcValT const *v);

    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.

3.8.16 Translation variant multi pixel operation

Pseudo code of the operation:

```
MultiPixOp(Dst, Src[], Mpo)
{
    foreach d, s, in Dst, Src[]
        Dst(d) = Mpo.doIt(Src[0](s) ... Src[N](s), s.x, s.y, s.z);
}
```

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

```
template<class DstValT, class SrcValT>
class MpoT
{
public:
    typedef HxTagTransVar       TransVarianceCategory;

                    MpoT(HxTagList& tags);

    DstValT         doIt(SrcValT const *v, int x, int y, int z);

    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.

3.8.17 M output, N input pixel operation

The pattern loops over all pixels in the N input images and applies the pixel functor to each pair of N pixels. The result set of M values is written to the corresponding positions in the M result image.

Instantiations of the pattern are invoked via

- **HxImageRep::MNPixOp** (p. 631)

Instantiation is done via the **HxImgFtorMNpo** (p. 804) functor, with the following template parameters:

- DstImgsSigT is the signature type of the destination images
- SrcImgsSigT is the signature type of the source images
- MNpoT is the type of the multi pixel functor

The requirements for the MNpoT parameter and an example of its use are given in the sections on variations:

- **Translation invariant M output, N input pixel operation** (p. 31)
- **Translation variant M output, N input pixel operation** (p. 32)

Tags for MNpoT variations:

- TransVarianceCategory : **HxTagTransVar** (p. 1204) or **HxTagTransInVar** (p. 1204)

For further reference :

- Method frame : **HxMfMNpo** (p. 1020)
- HxImageData member : **HxImageData::MNPixOp** (p. 594)

3.8.18 Translation invariant M output, N input pixel operation

Pseudo code of the operation:

```
MNPixOp(Dst[], Src[], MNpo)
{
    foreach d, s, in Dst[], Src[]
        Dst[0](d) ... Dst[M](d) = MNpo.doIt(Src[0](s), ..., Src[N](s));
}
```

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

```
template<class DstValT, class SrcValT>
class MNpoT
{
public:
    typedef HxTagTransInVar     TransVarianceCategory;

                    MNpoT(HxTagList& tags);

    void            doIt(DstValT *d, SrcValT const *s);

    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.

3.8.19 Translation variant M output, N input pixel operation

Pseudo code of the operation:

```
MNPixOp(Dst[], Src[], MNpo)
{
    foreach d, s, in Dst[], Src[]
        Dst[0](d) ... Dst[M](d) = MNpo.doIt(Src[0](s), ..., Src[N](s),
                                            i.x, i.y, i.z);
}
```

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

```
template<class DstValT, class SrcValT>
class MNpoT
{
public:
    typedef HxTagTransVar       TransVarianceCategory;

                    MNpoT(HxTagList& tags);

    void            doIt(DstValT *d, SrcValT const *s, int x, int y, int z);

    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.

3.8.20 In/Out pixel operation

The pattern loops (a number of times) over all pixels in the image and fetches/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.

Instantiations of the pattern are invoked via

- ”constructors/factory”
- **HxImageRep::exportOp** (p. 631)

Instantiation is done via the **HxImgFtorInOut** (p. 791) functor, with the following template parameters:

- ImgSigT is the signature type of the image
- InOutT is the type of the in/out pixel functor

The requirements for the InOutT parameter and an example of its use are given in the sections on variations:

- **Translation invariant, 1 phase pixel export operation** (p. 33)
- **Translation variant, 1 phase pixel export operation** (p. 34)
- **Translation invariant, 1 phase pixel import operation** (p. 34)

- **Translation variant, 1 phase pixel import operation** (p. 35)
- **Translation invariant, n phase pixel export operation** (p. 35)
- **Translation variant, n phase pixel export operation** (p. 36)
- **Translation invariant, n phase pixel import operation** (p. 36)
- **Translation variant, n phase pixel import operation** (p. 37)

Tags for InOutT variations:

- DirectionCategory : **HxTagPixOpIn** (p. 1201) or **HxTagPixOpOut** (p. 1201)
- TransVarianceCategory : **HxTagTransVar** (p. 1204) or **HxTagTransInVar** (p. 1204)
- PhaseCategory : **HxTag1Phase** (p. 1194) or **HxTagNPhase** (p. 1200)

For further reference :

- Method frame : <none>
- HxImageData member : **HxImageData::inout** (p. 591)

3.8.21 Translation invariant, 1 phase pixel export operation

Pseudo code of the operation:

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

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

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxTagPixOpOut   DirectionCategory;
    typedef HxTagTransInVar TransVarianceCategory;
    typedef HxTag1Phase     PhaseCategory;

                        PixOpT(HxTagList&);

    void                doIt(const ArithT& v);

    static HxString     className();
};
```

3.8.22 Translation variant, 1 phase pixel export operation

Pseudo code of the operation:

```
OutPixOp(In, PixOp)
{
    foreach i in In
        PixOp.doIt(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 HxTagPixOpOut   DirectionCategory;
    typedef HxTagTransVar   TransVarianceCategory;
    typedef HxTag1Phase     PhaseCategory;

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

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

    static HxString     className();
};
```

3.8.23 Translation invariant, 1 phase pixel import operation

Pseudo code of the operation:

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

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

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxTagPixOpIn    DirectionCategory;
    typedef HxTagTransInVar TransVarianceCategory;
    typedef HxTag1Phase     PhaseCategory;

                        PixOpT(HxTagList&);

    ArithT              doIt();

    static HxString     className();
};
```

3.8.24 Translation variant, 1 phase pixel import operation

Pseudo code of the operation:

```
InPixOp(Out, PixOp)
{
    foreach o in Out
        Out(o) = PixOp.doIt(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 HxTagPixOpIn    DirectionCategory;
    typedef HxTagTransVar   TransVarianceCategory;
    typedef HxTag1Phase     PhaseCategory;

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

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

    static HxString     className();
};
```

3.8.25 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.doIt(In(i));
        PixOp.done(p)
    }
}
```

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

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxTagPixOpOut   DirectionCategory;
    typedef HxTagTransInVar TransVarianceCategory;
    typedef HxTagNPhase     PhaseCategory;

                        PixOpT(HxTagList&);

    void                doIt(const ArithT& v);

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

    static HxString     className();
};
```

3.8.26 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.doIt(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 HxTagPixOpOut   DirectionCategory;
    typedef HxTagTransVar   TransVarianceCategory;
    typedef HxTagNPhase     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();
};
```

3.8.27 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.doIt();
        PixOp.done(p)
    }
}
```

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

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxTagPixOpIn    DirectionCategory;
    typedef HxTagTransInVar TransVarianceCategory;
    typedef HxTagNPhase     PhaseCategory;

                        PixOpT(HxTagList&);

    ArithT              doIt();

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

    static HxString     className();
};
```

3.8.28 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.doIt(o.x, o.y, o.z);
        PixOp.done(p)
    }
}
```

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

```
template<class ArithT>
class PixOpT
{
public:
    typedef HxTagPixOpIn    DirectionCategory;
    typedef HxTagTransVar   TransVarianceCategory;
    typedef HxTagNPhase     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();
};
```

3.8.29 Export operation with an extra image

The pattern loops over all pixels in the image and an extra image and offers the pixels to a pixel functor.

Instantiations of the pattern are invoked via

- **HxImageRep::exportOpExtra** (p. 632)

Instantiation is done via the **HxImgFtorExportExtra** (p. 708) functor, with the following template parameters:

- ImgSigT is the signature type of the image
- ExtraImgSigT is the signature type of the extra image
- ExportExtraT is the type of the export pixel functor

The requirements for the ExportExtraT parameter and an example of its use are given in the sections on variations:

- **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- **Translation variant, 1 phase export operation with an extra image** (p. 39)
- **Translation invariant, N phase export operation with an extra image** (p. 39)

- **Translation variant, N phase export operation with an extra image** (p. 40)

Tags for ExportExtraT variations:

- TransVarianceCategory : **HxTagTransVar** (p. 1204) or **HxTagTransInVar** (p. 1204)
- PhaseCategory : **HxTag1Phase** (p. 1194) or **HxTagNPhase** (p. 1200)

For further reference :

- Method frame : **HxMfExportExtra** (p. 1011)
- HxImageData member : **HxImageData::exportExtra** (p. 592)

3.8.30 Translation invariant, 1 phase export operation with an extra image

Pseudo code of the operation:

```
ExportOpExtra(In, Extra, ExportOp)
{
    foreach i, e in In, Extra
        ExportOp.doIt(In(i), Extra(e));
}
```

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

```
template<class ImValT, class ExtraValT>
class ExportExtraT
{
public:
    typedef HxTagTransInVar     TransVarianceCategory;
    typedef HxTag1Phase         PhaseCategory;

                        ExportExtraT(HxTagList&);

    void                doIt(const ImValT& imV, const ExtraValT& extraV);

    static HxString     className();
};
```

Example(s):

- instantiation : **HxInstExportExtraIdentMaskMean_c** (p. **??**), usage : **HxIdentMaskMean** (p. 239)

3.8.31 Translation variant, 1 phase export operation with an extra image

Pseudo code of the operation:

```
ExportOpExtra(In, Extra, ExportOp)
{
    foreach i, e in In, Extra
        ExportOp.doIt(In(i), Extra(e), i.x, i.y, i.z);
}
```

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

```
template<class ImValT, class ExtraValT>
class ExportExtraT
{
public:
    typedef HxTagTransVar       TransVarianceCategory;
    typedef HxTag1Phase         PhaseCategory;

                        ExportExtraT(HxTagList&);

    void                doIt(const ImValT& imV, const ExtraValT& extraV,
                             int x, int y, int z);

    static HxString     className();
};
```

Example(s):

- instantiation : **HxInstExportExtraIdentMaskMoments_c** (p. **??**), usage : **HxIdentMaskMoments** (p. 241)

3.8.32 Translation invariant, N phase export operation with an extra image

Pseudo code of the operation:

```
ExportOpExtra(In, Extra, ExportOp)
{
    for p = 1 to ExportOp.nrPhases();
    {
        ExportOp.init(p);
        foreach i, e in In, Extra
            ExportOp.doIt(In(i), Extra(e));
        ExportOp.done(p);
    }
}
```

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

```
template<class ImValT, class ExtraValT>
class ExportExtraT
{
public:
    typedef HxTagTransInVar     TransVarianceCategory;
    typedef HxTagNPhase         PhaseCategory;

                        ExportExtraT(HxTagList&);

    int                 nrPhases() const;
    void                init(int phase);
    void                doIt(const ImValT& imV, const ExtraValT& extraV);
    void                done(int phase);

    static HxString     className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.33 Translation variant, N phase export operation with an extra image

Pseudo code of the operation:

```
ExportOpExtra(In, Extra, ExportOp)
{
    for p = 1 to ExportOp.nrPhases();
    {
        ExportOp.init(p);
        foreach i, e in In, Extra
            ExportOp.doIt(In(i), Extra(e), i.x, i.y, i.z);
        ExportOp.done(p);
    }
}
```

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

```
template<class ImValT, class ExtraValT>
class ExportExtraT
{
public:
    typedef HxTagTransVar       TransVarianceCategory;
    typedef HxTagNPhase         PhaseCategory;

                        ExportExtraT(HxTagList&);

    int                 nrPhases() const;
    void                init(int phase);
    void                doIt(const ImValT& imV, const ExtraValT& extraV,
                             int x, int y, int z);
    void                done(int phase);

    static HxString     className();
};
```

Example(s):

- instantiation : **HxInstExportExtraIdentMaskCentralMoments_c** (p. **??**), usage : **HxIdentMaskCentralMoments** (p. 238)

3.8.34 Generalized convolution on 2D images

The pattern loops over all pixels in the image and performs a generalized convolution in the neighbourhood of each pixel. The result is written to the corresponding position in the result image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition). Using multiplication and addition as parameters in a generalized convolution thus turns it into a ”normal” convolution. However, the are other possibilities. For example, using addition and minimum results in an erosion.

Instantiations of the pattern are invoked via

- **HxImageRep::generalizedConvolution** (p. 632)

Instantiation is done via the **HxImgFtorGenConv2d** (p. 711) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

Example(s):

- instantiation : **HxInstGenConv2dMulAdd_c** (p. **??**), usage : **HxUniformNonSep** (p. 378)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::generalizedConvolution** (p. 594)

3.8.35 Generalized convolution on 2D images, separated by dimension

The pattern loops over all pixels in the image and performs a generalized convolution in the neighbourhood of each pixel, processing each dimension independent of the others. The result is written to the corresponding position in the result image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition). Using multiplication and addition as parameters in a generalized convolution thus turns it into a ”normal” convolution. However, the are other possibilities. For example, using addition and minimum results in an erosion.

Instantiations of the pattern are invoked via

- **HxImageRep::genConvSeparated** (p. 633)
- **HxImageRep::genConv2dSep** (p. 634)

Instantiation is done via the **HxImgFtorGenConv2dSep** (p. 717) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

Example(s):

- instantiation : **HxInstGenConv2dSepMulAdd_c** (p. **??**), usage : **HxCannyEdgeMap** (p. 94)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::genConv2dSep** (p. 597)

3.8.36 Generalized convolution on 2D images, with a 1D kernel

The pattern loops over all pixels in the image and performs a generalized convolution in the neighbourhood of each pixel, processing each dimension independent of the others. The result is written to the corresponding position in the result image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition). Using multiplication and addition as parameters in a generalized convolution thus turns it into a ”normal” convolution. However, the are other possibilities. For example, using addition and minimum results in an erosion.

Instantiations of the pattern are invoked via

- **HxImageRep::generalizedConvolutionK1d** (p. 633)
- **HxImageRep::genConvSeparated** (p. 633)
- **HxImageRep::genConv3dSep** (p. 634)

Instantiation is done via the **HxImgFtorGenConv2dK1d** (p. 714) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

Example(s):

- instantiation : **HxInstGenConv2dK1dMulAdd_c** (p. **??**), usage : **HxRecGauss** (p. 330)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::generalizedConvolutionK1d** (p. 595)
- HxImageData member : **HxImageData::genConvSeparated** (p. 596)

3.8.37 Generalized convolution on 3D images

The pattern loops over all pixels in the image and performs a generalized convolution in the neighbourhood of each pixel. The result is written to the corresponding position in the result image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition). Using multiplication and addition as parameters in a generalized convolution thus turns it into a ”normal” convolution. However, the are other possibilities. For example, using addition and minimum results in an erosion.

Instantiations of the pattern are invoked via

- **HxImageRep::generalizedConvolution** (p. 632)

Instantiation is done via the **HxImgFtorGenConv3d** (p. 721) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

Example(s):

- instantiation : **HxInstGenConv3dMulAdd_c** (p. **??**), usage : **HxUniformNonSep** (p. 378)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::generalizedConvolution** (p. 594)

3.8.38 Generalized convolution on 3D images, with a 1D kernel

The pattern loops over all pixels in the image and performs a generalized convolution in the neighbourhood of each pixel, processing each dimension independent of the others. The result is written to the corresponding position in the result image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition). Using multiplication and addition as parameters in a generalized convolution thus turns it into a ”normal” convolution. However, the are other possibilities. For example, using addition and minimum results in an erosion.

Instantiations of the pattern are invoked via

- **HxImageRep::generalizedConvolutionK1d** (p. 633)
- **HxImageRep::genConvSeparated** (p. 633)
- **HxImageRep::genConv3dSep** (p. 634)

Instantiation is done via the **HxImgFtorGenConv3dK1d** (p. 723) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

Example(s):

- instantiation : **HxInstGenConv3dK1dMulAdd_c** (p. **??**), usage : **HxUniform** (p. 377)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::generalizedConvolutionK1d** (p. 595)
- HxImageData member : **HxImageData::genConv3dSep** (p. 598)

3.8.39 Recursive generalized convolution on 2D images

The pattern loops over all pixels in the image and performs a recursive generalized convolution in the neighbourhood of each pixel. The result is written to the corresponding position in the same image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition).

Instantiations of the pattern are invoked via

- **HxImageRep::recGenConv** (p. 635)

Instantiation is done via the **HxImgFtorRecGenConv2d** (p. 828) functor, with the following template parameters:

- ImgSigT is the signature type of the source/destination image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor

Example(s):

- instantiation : **HxInstRecGenConv2dAddMin_c** (p. **??**), usage : **HxDistanceTransform** (p. 125)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::recGenConv** (p. 599)

3.8.40 Recursive generalized convolution on 2D images, with a 1D kernel

The pattern loops over all pixels in the image and performs a recursive generalized convolution in the neighbourhood of each pixel, processing each dimension independent of the others. The result is written to the corresponding position in the same image.

A generalized convolution is like a convolution but the multiplication and addition steps are replaced with functors. The functors are called gMul (for generalized multiplication) and gAdd (for generalized addition).

Instantiations of the pattern are invoked via

- **HxImageRep::recGenConv2dSep** (p. 635)

Instantiation is done via the **HxImgFtorRecGenConv2dK1d** (p. 830) functor, with the following template parameters:

- ImgSigT is the signature type of the source/destination image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor

Example(s):

- instantiation : **HxInstRecGenConv2dK1dMulAdd_c** (p. **??**), usage : **HxRecGauss** (p. 330)

For further reference :

- Method frame : **HxMfGenConv** (p. 1013)
- HxImageData member : **HxImageData::recGenConv2dSep** (p. 600)

3.8.41 Neighbourhood operation on 2D images

The pattern loops over all pixels in the image and applies the neighbourhood functor to each pixel. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::neighbourhoodOp**(HxString,HxTagList&) (p. 636)

Instantiation is done via the **HxImgFtorNgb2d** (p. 810) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- NgbT is the type of the neighbourhood functor

The requirements for the NgbT parameter and an example of its use are given in the sections on variations:

- **1 phase, coordinate enumerated neighbourhood operation** (p. 46)
- **1 phase, loop neighbourhood operation** (p. 47)
- **2 phase, loop neighbourhood operation** (p. 48)
- **N phase, loop neighbourhood operation** (p. 49)

Tags for NgbT variations:

- PhaseCategory : **HxTag1Phase** (p. 1194) or **HxTag2Phase** (p. 1195) or **HxTagNPhase** (p. 1200)
- IteratorCategory : **HxTagCnum** (p. 1195) or **HxTagLoop** (p. 1199)

For further reference :

- Method frame : **HxMfNgb** (p. 1026)
- HxImageData member : **HxImageData::neighbourhoodOp**(HxImageData*,HxString,HxTagList&) (p. 601)

3.8.42 1 phase, coordinate enumerated neighbourhood operation

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ngb)
{
    foreach d, s in Dst, Src {
        Ngb.init(s.x, s.y, Src(s));
        foreach n in Ngb.begin() .. Ngb.end()
            Ngb.next(n.x, n.y, Src(s + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT>
class NgbT
{
public:

    typedef HxTagCnum       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

    typedef HxCnum              CnumType;

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

    HxSizes                     size();

    CnumType                    begin();
    CnumType                    end();

    void                        init(int ix, int iy, SrcValT value);
    void                        next(int nx, int ny, SrcValT value);
    DstValT                     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);
};
```

Example(s):

- instantiation : **HxInstNgbNonMaxSuppression2d_c** (p. **??**), usage : **HxNonMaxSuppression-GradDir** (p. 305)
- instantiation : **HxInstNgbIsMaxGradDir2d_c** (p. **??**), usage : **HxCannyThresholdAlt** (p. 96)

3.8.43 1 phase, loop neighbourhood operation

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ngb)
{
    foreach d, s in Dst, Src {
        Ngb.init(s.x, s.y, Src(s));
        foreach n in Ngb.size()
            Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int ix, int iy, SrcValT value);
    void                        next(int nx, int ny, SrcValT value);
    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : **HxInstNgbPercentile2d_c** (p. **??**), usage : **HxPercentile** (p. 320)

3.8.44 2 phase, loop neighbourhood operation

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ngb)
{
    foreach d, s in Dst, Src {
        Ngb.init(s.x, s.y, Src(s));
        foreach n in Ngb.size()
            Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n));
        Ngb.init2(s.x, s.y, Src(s));
        foreach n in Ngb.size()
            Ngb.next2(n.x, n.y, Src(s - (Ngb.size()/2) + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag2Phase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int ix, int iy, SrcValT value);
```

```
    void                        next(int nx, int ny, SrcValT value);

    void                        init2(int ix, int iy, SrcValT value);
    void                        next2(int nx, int ny, SrcValT value);

    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.45 N phase, loop neighbourhood operation

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ngb)
{
    foreach d, s in Dst, Src {
        phase = 1;
        do {
            Ngb.init(phase, s.x, s.y, Src(s));
            foreach n in Ngb.size()
                Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n));
            Ngb.done(phase);
        } while (Ngb.hasNextPhase(phase++));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTagNPhase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int phase, int ix, int iy, SrcValT value);
    void                        next(int nx, int ny, SrcValT value);
    void                        done(int phase);
    bool                        hasNextPhase(int thisPhase);
    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.46 Neighbourhood operation on 2D images with an extra image

The pattern loops over all pixels in the image and the extra image and applies the neighbourhood functor to the pixels. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::neighbourhoodOpExtra**(HxString,HxImageRep,HxTagList&) (p. 636)

Instantiation is done via the **HxImgFtorNgb2dExtra** (p. 813) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- ExtraImgSigT is the signature type of the extra image
- NgbT is the type of the neighbourhood functor

The requirements for the NgbT parameter and an example of its use are given in the sections on variations:

- **1 phase, coordinate enumerated neighbourhood operation with an extra image** (p. 50)
- **1 phase, loop neighbourhood operation with an extra image** (p. 51)
- **2 phase, loop neighbourhood operation with an extra image** (p. 52)
- **N phase, loop neighbourhood operation with an extra image** (p. 53)

Tags for NgbT variations:

- PhaseCategory : **HxTag1Phase** (p. 1194) or **HxTag2Phase** (p. 1195) or **HxTagNPhase** (p. 1200)
- IteratorCategory : **HxTagCnum** (p. 1195) or **HxTagLoop** (p. 1199)

For further reference :

- Method frame : **HxMfNgb** (p. 1026)
- HxImageData member : **HxImageData::neighbourhoodOpExtra**(HxImageData∗,HxImageData∗,HxString,HxTagList&) (p. 601)

3.8.47 1 phase, coordinate enumerated neighbourhood operation with an extra image

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ngb)
{
    foreach d, s, e in Dst, Src, Ext {
        Ngb.init(s.x, s.y, Src(s), Ext(e));
        foreach n in Ngb.begin() .. Ngb.end()
            Ngb.next(n.x, n.y, Src(s + n), Ext(e + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT>
class NgbT
{
public:

    typedef HxTagCnum       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

    typedef HxCnum              CnumType;

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

    HxSizes                     size();

    CnumType                    begin();
    CnumType                    end();

    void                        init(int ix, int iy, SrcValT v1, ExtValT v2);
    void                        next(int nx, int ny, SrcValT v1, ExtValT v2);
    DstValT                     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);
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.48 1 phase, loop neighbourhood operation with an extra image

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ngb)
{
    foreach d, s, e in Dst, Src, Ext {
        Ngb.init(s.x, s.y, Src(s), Ext(e));
        foreach n in Ngb.size()
            Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                               Ext(e - (Ngb.size()/2) + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int ix, int iy, SrcValT v1, ExtValT v2);
    void                        next(int nx, int ny, SrcValT v1, ExtValT v2);
    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.49 2 phase, loop neighbourhood operation with an extra image

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ngb)
{
    foreach d, s, e in Dst, Src, Ext {
        Ngb.init(s.x, s.y, Src(s), Ext(e));
        foreach n in Ngb.size()
            Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                               Ext(e - (Ngb.size()/2) + n));
        Ngb.init2(s.x, s.y, Src(s), Ext(e));
        foreach n in Ngb.size()
            Ngb.next2(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                                Ext(e - (Ngb.size()/2) + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag2Phase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int ix, int iy, SrcValT v1, ExtValT v2);
    void                        next(int nx, int ny, SrcValT v1, ExtValT v2);
```

```
    void                        init2(int ix, int iy, SrcValT v1, ExtValT v2);
    void                        next2(int nx, int ny, SrcValT v1, ExtValT v2);

    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.50 N phase, loop neighbourhood operation with an extra image

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ngb)
{
    foreach d, s, e in Dst, Src, Ext {
        phase = 1;
        do {
            Ngb.init(phase, s.x, s.y, Src(s), Ext(e));
            foreach n in Ngb.size()
                Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                                   Ext(e - (Ngb.size()/2) + n));
            Ngb.done(phase);
        } while (Ngb.hasNextPhase(phase++));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTagNPhase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int phase, int ix, int iy, SrcValT v1,
                                     ExtValT v2);
    void                        next(int nx, int ny, SrcValT v1, ExtValT v2);
    void                        done(int phase);
    bool                        hasNextPhase(int thisPhase);
    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.51 Neighbourhood operation on 2D images with two extra images

The pattern loops over all pixels in the image and the extra images and applies the neighbourhood functor to the pixels. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::neighbourhoodOpExtra2**(HxString,HxImageRep,HxImageRep,HxTagList&) (p. 637)

Instantiation is done via the **HxImgFtorNgb2dExtra2** (p. 816) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- ExtraImgSigT is the signature type of the extra image
- ExtraImg2SigT is the signature type of second extra image
- NgbT is the type of the neighbourhood functor

The requirements for the NgbT parameter and an example of its use are given in the sections on variations:

- **1 phase, coordinate enumerated neighbourhood operation with two extra images** (p. 54)
- **1 phase, loop neighbourhood operation with two extra images** (p. 55)
- **2 phase, loop neighbourhood operation with two extra images** (p. 56)
- **N phase, loop neighbourhood operation with two extra images** (p. 57)

Tags for NgbT variations:

- PhaseCategory : **HxTag1Phase** (p. 1194) or **HxTag2Phase** (p. 1195) or **HxTagNPhase** (p. 1200)
- IteratorCategory : **HxTagCnum** (p. 1195) or **HxTagLoop** (p. 1199)

For further reference :

- Method frame : **HxMfNgb** (p. 1026)
- HxImageData member : **HxImageData::neighbourhoodOpExtra2**(HxImageData∗,HxImageData∗,HxImageData∗,HxString,HxTagList&) (p. 602)

3.8.52 1 phase, coordinate enumerated neighbourhood operation with two extra images

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ext2, Ngb)
{
    foreach d, s, e, e2 in Dst, Src, Ext, Ext2 {
        Ngb.init(s.x, s.y, Src(s), Ext(e), Ext2(e2));
        foreach n in Ngb.begin() .. Ngb.end()
            Ngb.next(n.x, n.y, Src(s + n), Ext(e + n), Ext2(e2 + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT, class Ext2ValT>
class NgbT
{
public:

    typedef HxTagCnum       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

    typedef HxCnum              CnumType;

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

    HxSizes           size();

    CnumType          begin();
    CnumType          end();

    void              init(int ix, int iy, SrcValT v1, ExtValT v2, Ext2ValT v3);
    void              next(int nx, int ny, SrcValT v1, ExtValT v2, Ext2ValT v3);
    DstValT           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);
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.53 1 phase, loop neighbourhood operation with two extra images

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ext2, Ngb)
{
    foreach d, s, e, e2 in Dst, Src, Ext, Ext2 {
        Ngb.init(s.x, s.y, Src(s), Ext(e), Ext2(e2));
        foreach n in Ngb.size()
            Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                               Ext(e - (Ngb.size()/2) + n),
                               Ext2(e2 - (Ngb.size()/2) + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT, class Ext2ValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

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

    HxSizes         size();

    void            init(int ix, int iy, SrcValT v1, ExtValT v2, Ext2ValT v3);
    void            next(int nx, int ny, SrcValT v1, ExtValT v2, Ext2ValT v3);
    DstValT         result() const;

    static HxString   className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.54 2 phase, loop neighbourhood operation with two extra images

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ext2, Ngb)
{
    foreach d, s, e, e2 in Dst, Src, Ext, Ext2 {
        Ngb.init(s.x, s.y, Src(s), Ext(e), Ext2(e2));
        foreach n in Ngb.size()
            Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                               Ext(e - (Ngb.size()/2) + n),
                               Ext2(e2 - (Ngb.size()/2) + n));
        Ngb.init2(s.x, s.y, Src(s), Ext(e), Ext2(e2));
        foreach n in Ngb.size()
            Ngb.next2(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                                Ext(e - (Ngb.size()/2) + n),
                                Ext2(e2 - (Ngb.size()/2) + n));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT, class Ext2ValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag2Phase     PhaseCategory;

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

```
    HxSizes           size();

    void              init(int ix, int iy, SrcValT v1, ExtValT v2, Ext2ValT v3);
    void              next(int nx, int ny, SrcValT v1, ExtValT v2, Ext2ValT v3);

    void              init2(int ix, int iy, SrcValT v1, ExtValT v2, Ext2ValT v3);
    void              next2(int nx, int ny, SrcValT v1, ExtValT v2, Ext2ValT v3);

    DstValT           result() const;

    static HxString   className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.55 N phase, loop neighbourhood operation with two extra images

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ext, Ext2, Ngb)
{
    foreach d, s, e, e2 in Dst, Src, Ext, Ext2 {
        phase = 1;
        do {
            Ngb.init(phase, s.x, s.y, Src(s), Ext(e), Ext2(e2));
            foreach n in Ngb.size()
                Ngb.next(n.x, n.y, Src(s - (Ngb.size()/2) + n),
                                   Ext(e - (Ngb.size()/2) + n),
                                   Ext2(e2 - (Ngb.size()/2) + n));
            Ngb.done(phase);
        } while (Ngb.hasNextPhase(phase++));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class ExtValT, class Ext2ValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTagNPhase     PhaseCategory;

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

    HxSizes           size();

    void              init(int phase, int ix, int iy, SrcValT v1, ExtValT v2,
                           Ext2ValT v3);
    void              next(int nx, int ny, SrcValT v1, ExtValT v2, Ext2ValT v3);
    void              done(int phase);
    bool              hasNextPhase(int thisPhase);
    DstValT           result() const;
```

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

Example(s):

- instantiation : <none>, usage : <none>

3.8.56 Kernel based neighbourhood operation on 2D images

The pattern loops over all pixels in the image and applies the neighbourhood functor to each pixel. The result is written to the corresponding position in the result image.

Instantiations of the pattern are invoked via

- **HxImageRep::neighbourhoodOp**(HxImageRep,HxString,HxTagList&) (p. 637)

Instantiation is done via the **HxImgFtorKernelNgb2d** (p. 795) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- NgbT is the type of the neighbourhood functor

The requirements for the NgbT parameter and an example of its use are given in the sections on variations:

- **1 phase, loop neighbourhood operation with kernel** (p. 58)
- **2 phase, loop neighbourhood operation using a kernel** (p. 59)
- **N phase, loop neighbourhood operation using a kernel** (p. 60)

Tags for NgbT variations:

- PhaseCategory : **HxTag1Phase** (p. 1194) or **HxTag2Phase** (p. 1195) or **HxTagNPhase** (p. 1200)
- IteratorCategory : **HxTagCnum** (p. 1195) or **HxTagLoop** (p. 1199)

For further reference :

- Method frame : **HxMfNgb** (p. 1026)
- HxImageData member : **HxImageData::neighbourhoodOp**(HxImageData∗,HxImageData∗,HxString,HxTagList&) (p. 603)

3.8.57 1 phase, loop neighbourhood operation with kernel

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ker, Ngb)
{
    foreach d, s in Dst, Src {
        Ngb.init(s.x, s.y, Src(s));
        foreach k in Ker
            Ngb.next(k.x, k.y, Src(s - (Ngb.size()/2) + k), Ker(k));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class KerValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag1Phase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int ix, int iy, SrcValT imVal);
    void                        next(int nx, int ny, SrcValT imVal, KerValT kerVal);
    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.58 2 phase, loop neighbourhood operation using a kernel

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ker, Ngb)
{
    foreach d, s in Dst, Src {
        Ngb.init(s.x, s.y, Src(s));
        foreach k in Ker
            Ngb.next(k.x, k.y, Src(s - (Ngb.size()/2) + k), Ker(k));
        Ngb.init2(s.x, s.y, Src(s));
        foreach k in Ker
            Ngb.next2(k.x, k.y, Src(s - (Ngb.size()/2) + k), Ker(k));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class KerValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTag2Phase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int ix, int iy, SrcValT imVal);
```

```
    void                        next(int nx, int ny, SrcValT imVal, KerValT kerVal);

    void                        init2(int ix, int iy, SrcValT imVal);
    void                        next2(int nx, int ny, SrcValT imVal, KerValT kerVal);

    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : **HxInstKerNgb2dNormCorrelation.c** (p. **??**), usage : **HxNormalizedCorrelation** (p. 307)

3.8.59 N phase, loop neighbourhood operation using a kernel

Pseudo code of the operation:

```
NeighbourhoodOp(Dst, Src, Ker, Ngb)
{
    foreach d, s in Dst, Src {
        phase = 1;
        do {
            Ngb.init(phase, s.x, s.y, Src(s));
            foreach k in Ker
                Ngb.next(k.x, k.y, Src(s - (Ngb.size()/2) + k), Ker(k));
            Ngb.done(phase);
        } while (Ngb.hasNextPhase(phase++));
        Dst(d) = ngb.result();
    }
}
```

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

```
template<class DstValT, class SrcValT, class KerValT>
class NgbT
{
public:

    typedef HxTagLoop       IteratorCategory;
    typedef HxTagNPhase     PhaseCategory;

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

    HxSizes                     size();

    void                        init(int phase, int ix, int iy, SrcValT imVal);
    void                        next(int nx, int ny, SrcValT imVal, KerValT kerVal);
    void                        done(int phase);
    bool                        hasNextPhase(int thisPhase);
    DstValT                     result() const;

    static HxString             className();
};
```

Example(s):

- instantiation : <none>, usage : <none>

3.8.60 Do It Yourself operation

The ”pattern” just offers the data pointers (see **Image data pointers** (p. 15)) to the functor.

Instantiations of the pattern are invoked via

- **HxImageRep::diyOp** (p. 638)

Instantiation is done via the **HxImgFtorDiy** (p. 705) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- DiyT is the type of the DIY functor

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

```
template<class DstDataPtrT, class SrcDataPtrT>
class DiyT
{
public:
                        DiyT(HxTagList&);

    void                doIt(DstDataPtrT dstPtr, SrcDataPtrT srcPtr,
                             HxSizes dstSize, HxSizes srcSize);

    static HxString     className();
};
```

Example(s):

- instantiation : **HxInstDiyTranspose_c** (p. **??**), usage : **HxTranspose** (p. 373)

For further reference :

- Method frame : **HxMfDiy** (p. 1009)
- HxImageData member : **HxImageData::diyOp** (p. 604)

3.8.61 Set operation

Instantiation is done via the **HxImgFtorSet** (p. 847) functor, with the following template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image

Example(s):

- instantiation : **HxInstantiatorSet_c** (p. **??**)

For further reference :

- HxImageData member : **HxImageData::setPartImage**(HxImageData∗,HxTagList&) (p. 592)

3.8.62 Border set for 2D images

Instantiation is done via the **HxImgFtorSetBorder2d** (p. 849) functor, with the following template parameters:

- ImgSigT is the signature type of the image

Example(s):

- instantiation : **HxImgFtorSetBorder2dInst_c** (p. **??**)

For further reference :

- HxImageData member : **HxImageData::setBorder**(HxTagList&) (p. 592)

3.8.63 Border set for 3D images

Instantiation is done via the **HxImgFtorSetBorder3d** (p. 852) functor, with the following template parameters:

- ImgSigT is the signature type of the image

Example(s):

- instantiation : **HxImgFtorSetBorder3dInst_c** (p. **??**)

For further reference :

- HxImageData member : **HxImageData::setBorder**(HxTagList&) (p. 592)

3.8.64 Geometric operation on 2D images

Instantiations of the pattern are invoked via

- **HxImageRep::geometricOp2d** (p. 638)

For further reference :

- Method frame : **HxMfResize** (p. 1031)
- HxImageData member : **HxImageData::geometricOp2d** (p. 605)

Note : operation is implemented by **HxImageTem2d::geometricOp2d** (p. 698)

Example(s):

- instantiation : <none>, usage : **HxReflect** (p. 332), **HxRotate** (p. 342), **HxScale** (p. 344)

3.8.65 Display RGB operation for 2D images

The pattern loops over all pixels in the image and converts the pixel value to an integer in ARGB formats for display purposes. The result is written to the given integer buffer.

Instantiations of the pattern are invoked via

- **HxImageRep::getRgbPixels2d**(int∗,HxString,int,int,HxGeoIntType) (p. 639)

Instantiation is done via the **HxImgFtorRgb2d** (p. 835) functor, with the following template parameters:

- ImgSigT is the signature type of the image
- RgbT is the type of the rgb pixel functor

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.

Example(s):

- instantiation : **HxInstRgb2dDirect_c** (p. **??**), usage :

For further reference :

- Method frame : <none>
- HxImageData member : **HxImageData::rgbOp** (p. 604)

3.8.66 Display RGB operation for 3D images

The pattern loops over all pixels in the image and converts the pixel value to an integer in ARGB formats for display purposes. The result is written to the given integer buffer.

Instantiations of the pattern are invoked via

- **HxImageRep::getRgbPixels3d** (p. 640)

Instantiation is done via the **HxImgFtorRgb3d** (p. 837) functor, with the following template parameters:

- ImgSigT is the signature type of the image
- RgbT is the type of the rgb pixel functor

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.

Example(s):

- instantiation : **HxInstRgb3dDirect_c** (p. **??**), usage :

For further reference :

- Method frame : <none>
- HxImageData member : **HxImageData::rgbOp** (p. 604)

Chapter 4

Detector

4.1 Detector

This chapter:

- **Histogram** (p. 65)
- **Segmentation** (p. 65)

4.2 Histogram

Classes:

HxHistogram (p. 548), HxHistoList

4.3 Segmentation

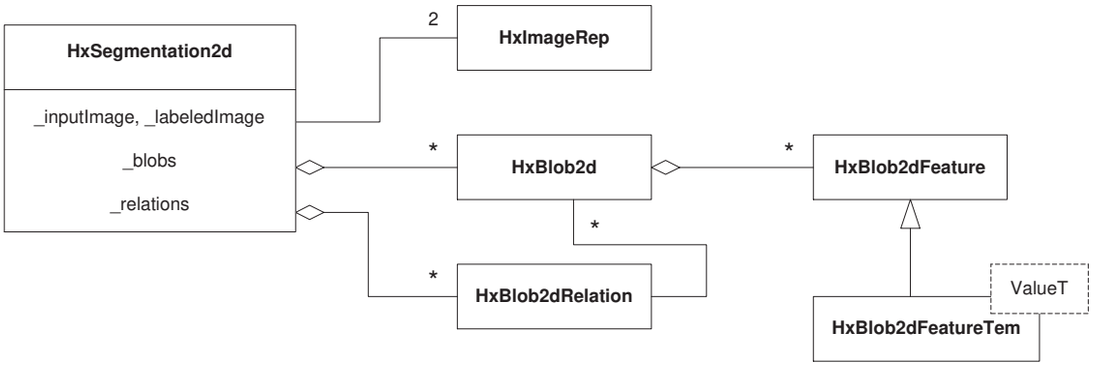


Figure 4.1: Class diagram HxSegmentation2d

Classes:

HxSegmentation2d (p. 1182), **HxBlob2d** (p. 402), **HxBlob2dRelation** (p. 411), **HxBlob2dList.h**, **HxBlob2dFeature** (p. 406), **HxBlob2dFeatureTem** (p. 408)

Global functions:

HxImageToSegmentation (p. 258), **HxSegmentationCentralMoments** (p. 344), **HxSegmentationHistogram** (p. 345), **HxSegmentationMean** (p. 346), **HxSegmentationMedian** (p. 346), **HxSegmentationMoments** (p. 347), **HxSegmentationStDev** (p. 347), **HxSegmentationSum** (p. 348), **HxSegmentationVariance** (p. 348)

Chapter 5

Geometry

5.1 BSplines

5.1.1 Terminology

5.1.1.1 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).

5.1.1.2 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.

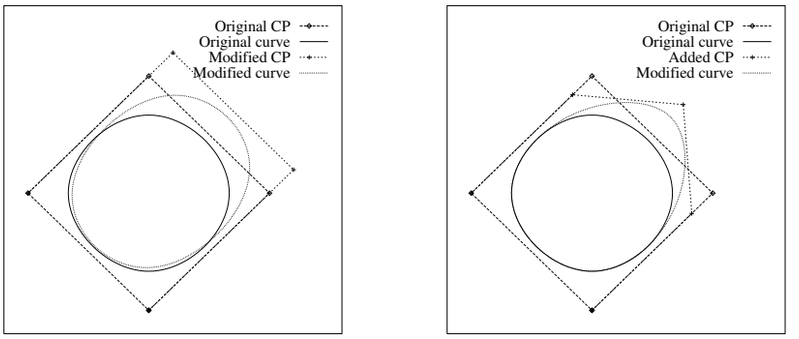


Figure 5.1: Examples of closed cubic B-Spline curves (degree=3). *Left:* Curve with four control points and uniformly spaced knots. Manipulation of one control point affects the whole curve. *Right:* A control point is added (knots become unevenly spaced). Manipulation of one control point affects only a curve interval.

5.1.1.3 B-Spline curves

We are actually interested in *B-spline curves* (see Figure 5.1), which are *splines* implemented as a linear combination of *basis functions* (see Figure 5.2).

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.

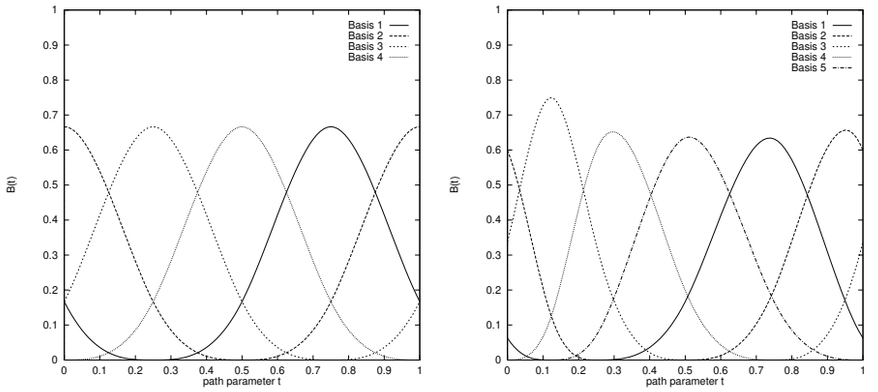


Figure 5.2: Basis functions for the curves in Figure 5.1. *Left:* Uniformly spaced knots. *Right:* Unevenly spaced knots.

5.1.2 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 ith control point for the computation of the nth derivative at path parameter t
- $B_{i,o}(t)$ = value of ith basis function of order o at path parameter t
- $B_{i,o}^{(n)}(t)$ = n^{th} derivative of ith 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:

$$x(t) = \sum_{i=l-d}^{l} X_i B_{i,d+1}(t)$$
$$y(t) = \sum_{i=l-d}^{l} Y_i B_{i,d+1}(t) \tag{5.1}$$

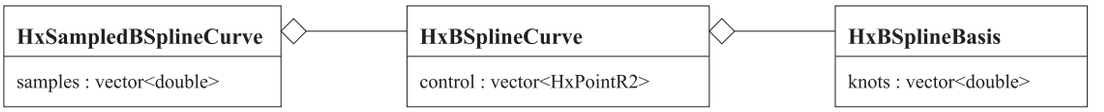


Figure 5.3: BSpline curve classes

The value of a ith 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 [**?**]):

$$\begin{aligned} B_{i,o}(t) &= \frac{t-\lambda_i}{\lambda_{i+o-1}-\lambda_i} B_{i,o-1}(t) + \frac{\lambda_{i+o}-t}{\lambda_{i+o}-\lambda_{i+1}}, B_{i+1,o-1}(t) \\ B_{i,1}(t) &= \begin{cases} 1, & \text{if } t \in [\lambda_i, \lambda_{i+1}), \\ 0, & \text{if } t \notin [\lambda_i, \lambda_{i+1}). \end{cases} \end{aligned} \tag{5.2}$$

The nth 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 [**?**]):

$$C^{(n)}(t) = \prod_{i=1}^{n}(d+1-l) \sum_{i=l-d+n}^{i} P_i^{(n)} B_{i,d+1-n}(t) \tag{5.3}$$

with

$$P_i^{(n)} = \begin{cases} P_i, & \text{if } n = 0, \\ \frac{P_i^{(n-1)} - P_{i-1}^{(n-1)}}{\lambda_{l+d+1-n}-\lambda_l}, & \text{if } n > 0. \end{cases} \tag{5.4}$$

The value of the nth derivative of the ith 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:

$$\begin{aligned} B_{i,o}^{(n)}(t) &= (o-1)\left\{ \frac{B_{i,o-1}^{(n-1)}(t)}{\lambda_{i+o-1}-\lambda_i} - \frac{B_{i+1,o-1}^{(n-1)}(t)}{\lambda_{i+o}-\lambda_{i+1}} \right\} \\ B_{i,o}^{(0)}(t) &= B_{i,o}(t) \end{aligned} \tag{5.5}$$

5.1.3 Class definition

The class design for the implementation is given in Figure 5.3.

5.1.3.1 `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.

5.1.3.2 `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 $t \in [minT, maxT)$.

5.1.3.3 `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 6

Registry

6.1 Registry

This chapter:

- **Overview of the registry** (p. 72)
- **Creating a registry** (p. 73)
- **Navigating a registry** (p. 73)

6.1.1 Overview of the registry

The registry (**HxRegistry** (p. 1088)) is a structured set of named values.

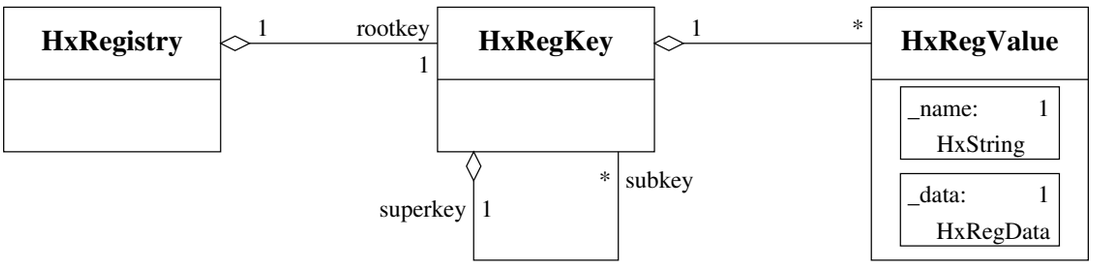


Figure 6.1: Class diagram HxRegistry

A registry value (represented by **HxRegValue** (p. 1104)) is a pair of a name (represented by HxString) and a data object (represented by **HxRegData** (p. 1084)). The data of a value is an integer or a string. Values are grouped in sets, each set is associated with a key (**HxRegKey** (p. 1097)). Keys are structured in the sense that keys have sub keys and form a tree structure. The uniquely identifying name of a key is determined by its path from the root key of the containing tree. In a key name the components of its path are separated by "/". A registry contains one tree of keys, shown in the class diagram as a 1-1 relationship between classes **HxRegistry** (p. 1088) and **HxRegKey** (p. 1097).

An example of the contents of a registry:

```
[/methods/getName]
```

```
"name"="getName"
"nArgs"=0
[/methods/setName]
"name"="setName"
"nArgs"=1
"type0"="string"
```

Note that each value is uniquely identified by the path of its containing key and its name. A tree of registry keys and their associated values has its analogy in a directory tree with files. Keys (directories) can contain both keys as sub keys (subdirectories) and values (files). Both keys and values have a name that is not necessarily unique. Both have a path that uniquely identifies them within the key tree.

6.1.2 Creating a registry

```
class HxRegistry {
                        HxRegistry();
                        ~HxRegistry();

    static HxRegistry&  instance();

    HxRegKey*           insertKey(HxString path);
    void                eraseKey(HxString path);
};

HxRegistry::instance().insertKey("/methods/getCount");
```

The class **HxRegistry** (p. 1088) provides a default constructor to create a new empty registry. It also grants access to a unique instance of the class. In this respect the class **HxRegistry** (p. 1088) behaves as a singleton class. This unique instance can be used to share a registry throughout the complete library and possible applications. In the Horus library, this instance is the "horus registry".

Keys can be inserted with the method **HxRegistry::insertKey** (p. 1092). The key to be inserted must be specified with its path. Paths are absolute (from the root of the registry), or relative to the "current" key in the registry. Absolute paths start with "/". If a specified key already exists, nothing happens. See below for a description of the current key.

6.1.3 Navigating a registry

A registry can be navigated in two ways. One way is to only use the interface provided by the class **HxRegistry** (p. 1088). The other way is to obtain and manipulate the individual keys of a registry.

Insert and find values

```
class HxRegistry {
    void        insertValue(HxString path, const HxRegData&);
    void        eraseValue(HxString path);

    HxString    findValue(HxString path) const;
    int         valueExists(HxString path) const;
};

HxRegistry::instance().insertValue("/methods/getCount/nArgs", HxRegData(0));
```

If the class interface of **HxRegistry** (p. 1088) is used, values must be specified by path. Again, this can be absolute from the registry root or relative to the current or cursor key of the registry. Note that the result of findValue is always the string representation of the data part of the value. If the value does not exist an

empty string is returned. Since an empty string is a legal data part of a value, the method valueExists is provided to determine the existence of a value.

The cursor key

```
class HxRegistry {
    HxRegKey*   setCursorKey(HxString path);
    HxRegKey*   setCursorUp();
    HxRegKey*   getCursorKey() const;
};
```

Accessing values and keys relative from the cursor key is more efficient since the specified path does not have to be walked down completely from the root for each access. Some methods are provided to manipulate the cursor key.

Accessing values using the methods provided by the class **HxRegistry** (p. 1088) is neither the most efficient nor the most powerful way to manipulate the values of a key. A better way to manipulate the values of a key is using methods provided by the class **HxRegKey** (p. 1097).

Managing the values of a key

```
class HxRegKey {
    void                insertValue(const HxRegValue& value);
    void                eraseValue(HxString name);

    const HxRegValue*   findValue(HxString name) const;
};
```

To obtain a specific key, use the method **HxRegistry::findKey** (p. 1093). A pointer to a key is returned that can be used to manipulate the values of that key or to navigate further in the key tree.

To obtain a pointer to a value use the method **HxRegKey::findValue** (p. 1102) on its containing key. If a value with the specified name does not exist, a null pointer is returned. A registry value contains an object of class **HxRegData** (p. 1084). This object is either a string or an integer. The method **HxRegData::type** (p. 1087) is provided to examine the type of a **HxRegData** (p. 1084) object. With methods **HxRegData::getInt** (p. 1088) and **HxRegData::getString** (p. 1087)} the integer or string value are obtained.

Manipulating a value

```
    int nItems = -1;
    HxRegValue* val = regKey->findValue("nItems");
    if (val) {
        HxRegData data = val->getData();
        if (data.type() == HxRegData::Int) {
            nItems = data.getInt();
        }
    }
```

Chapter 7

Global Image Functions Reference

7.1 HxAbs.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxAbs** (**HxImageRep** im)
  Absolute value.

7.1.1 Detailed Description

7.1.2 Function Documentation

7.1.2.1 HxImageRep L_HXIMAGEREP HxAbs (HxImageRep *im*)

Absolute value.

The function computes the absolute value (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics :

- Pattern : **Unary pixel operation** (p. 26)
- Variation : **Translation invariant unary pixel operation** (p. 26)
- The pixel functor : **HxUpoAbs** (p. 1205)
- Instantiations : **HxInstantiatorAbs_c** (p. **??**)

```
13 {
14     HxString fname("HxAbs");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     return im.unaryPixOp("abs");
22 }
```

7.2 HxAcos.h File Reference

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

Functions

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

  Arc cosine.

7.2.1 Detailed Description

7.2.2 Function Documentation

7.2.2.1 HxImageRep L_HXIMAGEREP HxAcos (HxImageRep *im*)

Arc cosine.

The function computes the arc cosine (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoAcos** (p. 1206). The image functor instantiator : **HxInstantiatorAcos** (p. 864).

```
15 {
16     HxString fname("HxAcos");
17 
18     if (im.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     // value between -1 and 1
24     // ook voor vector etc.
25 
26     HxValue vsup = HxPixSup(im);
27     HxValue vinf = HxPixInf(im);
28 
29     if (im.signature().pixelDimensionality() == 1)
30     {
31         if ((((HxScalarDouble) vinf) < -1.0) || (((HxScalarDouble) vsup) > 1.0))
32         {
33             HxGlobalError::instance()->reportError(fname, im.name(), "values not in [-1:1]", HxGlobalError::HX_GE_OU
34         }
35     }
36     else if (im.signature().pixelDimensionality() == 2)
37     {
38         HxVec2Double vinf2d = (HxVec2Double) vinf;
39         HxVec2Double vsup2d = (HxVec2Double) vsup;
40         if ((vinf2d.x() < -1.0) || (vsup2d.x() > 1.0) ||
41             (vinf2d.y() < -1.0) || (vsup2d.y() > 1.0))
42         {
43             HxGlobalError::instance()->reportError(fname, im.name(), "2D values are not in [-1:1]", HxGlobalError::HX
44         }
45     }
46     else if (im.signature().pixelDimensionality() == 3)
47     {
48         HxVec3Double vinf3d = (HxVec3Double) vinf;
```

```
49         HxVec3Double vsup3d = (HxVec3Double) vsup;
50         if ((vinf3d.x() < -1.0) || (vsup3d.x() > 1.0) ||
51             (vinf3d.y() < -1.0) || (vsup3d.y() > 1.0) ||
52             (vinf3d.z() < -1.0) || (vsup3d.z() > 1.0))
53         {
54             HxGlobalError::instance()->reportError(fname, im.name(), "3D values are not in [-1:1]", HxGlobalError::HX
55         }
56     }
57
58     return im.unaryPixOp("acos");
59 }
```

7.3 HxAdd.h File Reference

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

Functions

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

  Addition.

7.3.1 Detailed Description

7.3.2 Function Documentation

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

Addition.

The function performs addition (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoAdd** (p. 417). The image functor instantiator : **HxInstantiatorAdd** (p. 865).

```
16 {
17     HxString fname("HxAdd");
18
19     if (im1.isNull())
20     {
21         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
22         return HxImageRep();
23     }
24     if (im2.isNull())
25     {
26         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
27         return HxImageRep();
28     }
29
30     if (im1.dimensionality() != im2.dimensionality())
31     {
32         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
33         return HxImageRep();
34     }
35     if (im1.pixelDimensionality() != im2.pixelDimensionality())
```

```
36     {
37         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
38         return HxImageRep();
39     }
40
41     if (im1.sizes().x() != im2.sizes().x())
42     {
43         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
44         return HxImageRep();
45     }
46     if (im1.sizes().y() != im2.sizes().y())
47     {
48         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
49         return HxImageRep();
50     }
51     if (im1.dimensionality() > 2)
52     {
53         if (im1.sizes().z() != im2.sizes().z())
54         {
55             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGE
56             return HxImageRep();
57         }
58     }
59
60     // in case of byte, unsigned: generate warnings in case of potentially dangerous
61     // situations.
62     // Check if image is byte.
63
64     if (((im1.signature().pixelType() == INT_VALUE) &&
65          (im1.signature().pixelPrecision() == 8)) ||
66         ((im2.signature().pixelType() == INT_VALUE) &&
67          (im2.signature().pixelPrecision() == 8)))
68     {
69         if ((im1.pixelDimensionality() == 1) && (im1.pixelDimensionality() == 1))
70         {
71             if ((HxPixMax(im1).HxScalarIntValue() +
72                  HxPixMax(im2).HxScalarIntValue()) > HxScalarInt(255))
73             {
74                 HxGlobalError::instance()->reportWarning(fname,
75                     im1.name()+HxString(" ")+im2.name(),
76                     "possible overflow due to byte precision",
77                     HxGlobalError::HX_GW_OVERFLOW);
78             }
79         }
80         else if ((HxPixMax(HxUnaryMax(im1)).HxScalarIntValue() +
81                   HxPixMax(HxUnaryMax(im2)).HxScalarIntValue()) > HxScalarInt(255))
82         {
83                 HxGlobalError::instance()->reportWarning(fname,
84                     im1.name()+HxString(" ")+im2.name(),
85                     "possible overflow due to byte precision",
86                     HxGlobalError::HX_GW_OVERFLOW);
87         }
88     }
89
90     return im1.binaryPixOp(im2, "add");
91 }
```

7.4 HxAddBinaryNoise.h File Reference

Functions

- **HxImageRep** L_HXIMAGEREP **HxAddBinaryNoise** (**HxImageRep** im, double percent=0.2)

Add binary **noise** (p. **??**) *to an image.*

7.4.1 Detailed Description

7.4.2 Function Documentation

7.4.2.1 HxImageRep L_HXIMAGEREP HxAddBinaryNoise (HxImageRep *im*, double *percent* = 0.2)

Add binary **noise** (p. **??**) to an image.

Numerical Recipes in C, 2nd edition, Cambridge University Press, Cambridge, 1992.

```
28 {
29     HxSizes sz = im.sizes();
30     HxTagList   tags;
31     HxAddTag(tags, "size", sz);
32     HxAddTag(tags, "seed", -3);
33     HxAddTag(tags, "percent", percent);
34
35     HxImageRep noiseIm =  HxImageFactory::instance().fromNamedGenerator(
36             HXIMAGESIG2DDOUBLE, "binaryNoise", tags);
37
38     noiseIm = HxMulVal(noiseIm, HxPixMax(im));
39
40     return HxAdd(noiseIm,im);
41
42 //  return HxMul(im, noiseIm);
43 }
```

7.5 HxAddGaussianNoise.h File Reference

Functions

- **HxImageRep** L_HXIMAGEREP **HxAddGaussianNoise** (**HxImageRep** im, double mean, double stdev)

  Add Gaussian **noise** (p. **??**) *to an image.*

7.5.1 Detailed Description

7.5.2 Function Documentation

7.5.2.1 HxImageRep L_HXIMAGEREP HxAddGaussianNoise (HxImageRep *im*, double *mean*, double *stdev*)

Add Gaussian **noise** (p. **??**) to an image.

Numerical Recipes in C, 2nd edition, Cambridge University Press, Cambridge, 1992.

```
28 {
29     HxSizes sz = im.sizes();
30     HxTagList   tags;
```

```
31     HxAddTag(tags, "size", sz);
32     HxAddTag(tags, "seed", -3);
33     HxAddTag(tags, "mean", mean);
34     HxAddTag(tags, "stdev", stdev);
35
36     HxImageRep noiseIm =  HxImageFactory::instance().fromNamedGenerator(
37             HXIMAGESIG2DDOUBLE, "gaussianNoise", tags);
38
39     noiseIm = HxMulVal(noiseIm, HxPixMax(im));
40
41     return HxAdd(noiseIm,im);
42
43 //  return HxMul(im, noiseIm);
44 }
```

7.6 HxAddPoissonNoise.h File Reference

Functions

- **HxImageRep** L_HXIMAGEREP **HxAddPoissonNoise** (**HxImageRep** im, double conversionFactor)

  Add Poisson **noise** (p. **??**) *to an image.*

7.6.1 Detailed Description

7.6.2 Function Documentation

7.6.2.1 HxImageRep L_HXIMAGEREP HxAddPoissonNoise (HxImageRep *im*, double *conversionFactor*)

Add Poisson **noise** (p. **??**) to an image.

Numerical Recipes in C, 2nd edition, Cambridge University Press, Cambridge, 1992.

```
28 {
29     HxSizes sz = im.sizes();
30     HxTagList   tags;
31     HxAddTag(tags, "size", sz);
32     HxAddTag(tags, "seed", -3);
33     HxAddTag(tags, "mean", conversionFactor);
34
35     HxImageRep noiseIm =  HxImageFactory::instance().fromNamedGenerator(
36             HXIMAGESIG2DDOUBLE, "poissonNoise", tags);
37
38     noiseIm = HxMulVal(noiseIm, HxPixMax(im));
39
40     return HxAdd(noiseIm,im);
41
42 //  return HxMul(im, noiseIm);
43 }
```

7.7 HxAddSat.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxAddSat** (**HxImageRep** im1, **HxImageRep** im2)
  Saturated addition.

7.7.1 Detailed Description

7.7.2 Function Documentation

7.7.2.1 HxImageRep L_HXIMAGEREP HxAddSat (HxImageRep *im1*, HxImageRep *im2*)

Saturated addition.

The function computes the saturated addition of all corresponding pixels in the input images via a binary pixel operation.

```
130 {
131     // call HxImageRep member function to do the image processing
132 //  std::cout << im1.signature() << std::endl;
133 //  std::cout << im2.signature() << std::endl;
134
135     int val = HxPixMax(im1).HxScalarIntValue().x();
136     if(abs(val-255) < 10)
137         val = 255;
138     if(abs(val-65535) < 10)
139         val = 65535;
140
141
142     HxTagList tl;
143     HxAddTag(tl,"maxSat", val );
144
145
146     return im1.binaryPixOp(im2, "addSat", tl);
147 }
```

7.8 HxAddUniformNoise.h File Reference

Functions

- **HxImageRep** L_HXIMAGEREP **HxAddUniformNoise** (**HxImageRep** im)
  Add uniform **noise** (p. **??**) *to an image.*

7.8.1 Detailed Description

7.8.2 Function Documentation

7.8.2.1 HxImageRep L_HXIMAGEREP HxAddUniformNoise (HxImageRep *im*)

Add uniform **noise** (p. **??**) to an image.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. Numerical Recipes in C, 2nd edition, Cambridge University Press, Cambridge, 1992.

```
16 {
17     HxSizes sz = im.sizes();
18     HxTagList   tags;
19     HxAddTag(tags, "size", sz);
20     HxAddTag(tags, "seed", -3);
21
22     HxImageRep noiseIm =  HxImageFactory::instance().fromNamedGenerator(
23             HXIMAGESIG2DDOUBLE, "uniformNoise", tags);
24     //the noise image has values in the range (0,1)
25 //  double maxVal = HxPixMax(im);
26     noiseIm = HxMulVal(noiseIm, HxPixMax(im));
27
28     return HxAdd(noiseIm,im);
29
30 //  return HxMul(im, noiseIm);
31
32 }
```

7.9 HxAddVal.h File Reference

`#include "HxImageRep.h"`

Functions

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

  Addition.

7.9.1 Detailed Description

7.9.2 Function Documentation

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

Addition.

The function performs addition (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoAdd** (p. 417). The image functor instantiator : **HxInstantiatorAddV** (p. 867).

```
14 {
15     HxString fname("HxAddVal");
16
17     if (im.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22
23     int valdim;
24     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
25     {
26         valdim = 1;
27     }
```

```
28     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
29     {
30         valdim = 2;
31     }
32     else
33     {
34         valdim = 3;
35     }
36     if (im.signature().pixelDimensionality() != valdim)
37     {
38         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
39                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
40         return HxImageRep();
41     }
42
43     return im.binaryPixOp(val, "add");
44 }
```

7.10 HxAffinePix.h File Reference

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

Functions

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

7.10.1 Detailed Description

7.11 HxAlternateSequentialFilter.h File Reference

```
#include "HxSF.h"
```

Typedefs

- typedef enum **OC** = 1
- typedef enum **CO**
- typedef enum **OCO**
- typedef enum COC **opSeq**

Functions

- **HxImageRep** L_HXIMAGEREP **HxAlternateSequentialFilter** (**HxImageRep** im, **HxSF** sf, int nrIter=1, opSeq s=OC)

  function y=mmasf_equ(f, SEQ, b, n)
  switch SEQ
  case 'OC',
  y = f;
  for i=1:n
  nb = mmsesum(b,i);
  y = mmclose(y,nb);

y = mmopen(y,nb);
end;
case 'CO',
y = f;
for i=1:n
nb = mmsesum(b,i);
y = mmopen(y,nb);
y = mmclose(y,nb);
end;
case 'OCO',
y = f;
for i=1:n
nb = mmsesum(b,i);
y = mmopen(y,nb);
y = mmclose(y,nb);
y = mmopen(y,nb);
end;
case 'COC', y = f; for i=1:n nb = mmsesum(b,i); y = mmclose(y,nb); y = mmopen(y,nb); y = mmclose(y,nb);
end; end;.

7.11.1 Detailed Description

7.11.2 Function Documentation

7.11.2.1 HxImageRep L_HXIMAGEREP HxAlternateSequentialFilter (HxImageRep *im*, HxSF *sf*, int *nrIter* = 1, opSeq *s* = OC)

function y=mmasf_equ(f, SEQ, b, n)

switch SEQ

case 'OC',

y = f;

for i=1:n

nb = mmsesum(b,i);

y = mmclose(y,nb);

y = mmopen(y,nb);

end;

case 'CO',

y = f;

for i=1:n

nb = mmsesum(b,i);

y = mmopen(y,nb);

y = mmclose(y,nb);

end;

case 'OCO',

y = f;

for i=1:n

nb = mmsesum(b,i);

y = mmopen(y,nb);

y = mmclose(y,nb);

y = mmopen(y,nb);

end;

case 'COC', y = f; for i=1:n nb = mmsesum(b,i); y = mmclose(y,nb); y = mmopen(y,nb); y = mmclose(y,nb); end; end;.

where, mmsesum(b,n) is n-1 iterative Minkowski additions

```
20 {
21     HxImageRep res=im;
22     HxSF sfn = sf;
23     int i;
24
25     switch(s){
26     case OC:
27             for(i=0; i< nrIter; i++)
28             {
29                 sfn = sf.dilateSF(i);
30                 res = HxClosing(res, sfn);
31                 res = HxOpening(res, sfn);
32             }
33             break;
34     case CO:
35             for(i=0; i< nrIter; i++)
36             {
37                 sfn = sf.dilateSF(i);
38                 res = HxOpening(res, sfn);
39                 res = HxClosing(res, sfn);
40             }
41             break;
42     case OCO:
43             for(i=0; i< nrIter; i++)
44             {
45                 sfn = sf.dilateSF(i);
46                 res = HxClosing(res, sfn);
47                 res = HxOpening(res, sfn);
48                 res = HxClosing(res, sfn);
49             }
50             break;
51     case COC:
52             for(i=0; i< nrIter; i++)
53             {
54                 sfn = sf.dilateSF(i);
55                 res = HxOpening(res, sfn);
56                 res = HxClosing(res, sfn);
57                 res = HxOpening(res, sfn);
58             }
59             break;
60     }
61
62
63     return res;
64 }
```

7.12 HxAnd.h File Reference

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

Functions

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

  And.

7.12.1 Detailed Description

7.12.2 Function Documentation

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

And.

The function performs and (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoAnd** (p. 422). The image functor instantiator : **HxInstantiatorAnd** (p. 868).

```
13 {
14     HxString fname("HxAnd");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.signature().imageDimensionality() != im2.signature().imageDimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32     if (im1.signature().pixelDimensionality() != im2.signature().pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
35         return HxImageRep();
36     }
37     if (im1.signature().pixelDimensionality() != 1)
38     {
39         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobal
40         return HxImageRep();
41     }
42 
43     if (im1.signature().pixelType() != im2.signature().pixelType())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
```

```
47     }
48     if (im1.signature().pixelType() != INT_VALUE)
49     {
50         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer values", HxGlobalE
51         return HxImageRep();
52     }
53
54     if (im1.signature().pixelPrecision() != im2.signature().pixelPrecision())
55     {
56         HxGlobalError::instance()->reportError(fname, "unequal pixel precisions", HxGlobalError::HX_GE_UNEQUAL_IMAGES
57         return HxImageRep();
58     }
59
60     if (im1.sizes().x() != im2.sizes().x())
61     {
62         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
63         return HxImageRep();
64     }
65     if (im1.sizes().y() != im2.sizes().y())
66     {
67         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
68         return HxImageRep();
69     }
70     if (im1.signature().imageDimensionality() > 2)
71     {
72         if (im1.sizes().z() != im2.sizes().z())
73         {
74             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
75             return HxImageRep();
76         }
77     }
78
79     return im1.binaryPixOp(im2, "and");
80 }
```

7.13 HxAndVal.h File Reference

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

Functions

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

7.13.1 Detailed Description

7.13.2 Function Documentation

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

And.

The function performs and (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoAnd** (p. 422). The image functor instantiator : **HxInstantiatorAndV** (p. 868).

```
13 {
14     HxString fname("HxAndVal");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21 
22     if (im.signature().pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
25         return HxImageRep();
26     }
27 
28     if (im.signature().pixelType() != INT_VALUE)
29     {
30         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
31         return HxImageRep();
32     }
33 
34     if (val.tag() != HxValue::SI)
35     {
36         HxGlobalError::instance()->reportError(fname, "only scalar integer value supported", HxGlobalError::HX_GE_INV
37         return HxImageRep();
38     }
39 
40 
41     return im.binaryPixOp(val, "and");
42 }
```

7.14 HxAreaClosing.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAreaClosing** (**HxImageRep** im, int conn, int minarea)

7.14.1 Detailed Description

7.15 HxAreaOpening.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxAreaOpening** (**HxImageRep** im, int conn, int area)

7.15.1 Detailed Description

7.16 HxArg.h File Reference

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

Functions

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

7.16.1 Detailed Description

7.16.2 Function Documentation

7.16.2.1 HxImageRep L_HXIMAGEREP HxArg (HxImageRep *im*)

Argument.

The function computes the complex argument (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoArg** (p. 1208). The image functor instantiator : **HxInstantiatorArg** (p. 869).

```
13 {
14     HxString fname("HxArg");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     if (im.signature().pixelType() != COMPLEX_VALUE)
23     {
24         HxGlobalError::instance()->reportError(fname, "Operation is only valid on complex images", HxGlobalError::HX_
25         return HxImageRep();
26     }
27
28     return im.unaryPixOp("arg");
29 }
```

7.17 HxAsin.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxAsin** (**HxImageRep** im)
  Arc sine.

7.17.1 Detailed Description

7.17.2 Function Documentation

7.17.2.1 HxImageRep L_HXIMAGEREP HxAsin (HxImageRep *im*)

Arc sine.

The function computes the arc sine (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoAsin** (p. 1209). The image functor instantiator : **HxInstantiatorAsin** (p. 869).

```
13 {
14     HxString fname("HxAsin");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     // value between -1 and 1
22     // ook voor vector etc.
23
24     HxValue vsup = HxPixSup(im);
25     HxValue vinf = HxPixInf(im);
26
27     if (im.signature().pixelDimensionality() == 1)
28     {
29         if ((((HxScalarDouble) vinf) < -1.0) || (((HxScalarDouble) vsup) > 1.0))
30         {
31             HxGlobalError::instance()->reportError(fname, im.name(), "values not in [-1:1]", HxGlobalError::HX_GE_OU
32         }
33     }
34     else if (im.signature().pixelDimensionality() == 2)
35     {
36         HxVec2Double vinf2d = (HxVec2Double) vinf;
37         HxVec2Double vsup2d = (HxVec2Double) vsup;
38         if ((vinf2d.x() < -1.0) || (vsup2d.x() > 1.0) ||
39             (vinf2d.y() < -1.0) || (vsup2d.y() > 1.0))
40         {
41             HxGlobalError::instance()->reportError(fname, im.name(), "2D values are not in [-1:1]", HxGlobalError::HX
42         }
43     }
44     else if (im.signature().pixelDimensionality() == 3)
45     {
46         HxVec3Double vinf3d = (HxVec3Double) vinf;
47         HxVec3Double vsup3d = (HxVec3Double) vsup;
48         if ((vinf3d.x() < -1.0) || (vsup3d.x() > 1.0) ||
49             (vinf3d.y() < -1.0) || (vsup3d.y() > 1.0) ||
50             (vinf3d.z() < -1.0) || (vsup3d.z() > 1.0))
51         {
52             HxGlobalError::instance()->reportError(fname, im.name(), "3D values are not in [-1:1]", HxGlobalError::HX
53         }
54     }
55
56     return im.unaryPixOp("asin");
57 }
```

7.18 HxAtan.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxAtan** (**HxImageRep** im)

  Arc tangent.

7.18.1 Detailed Description

7.18.2 Function Documentation

7.18.2.1 HxImageRep L_HXIMAGEREP HxAtan (HxImageRep *im*)

Arc tangent.

The function computes the arc tangent (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoAtan** (p. 1211). The image functor instantiator : **HxInstantiatorAtan** (p. 870).

```
13 {
14     HxString fname("HxAtan");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21 
22     return im.unaryPixOp("atan");
23 }
```

7.19 HxAtan2.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxAtan2** (**HxImageRep** im)

  Arc tangent (use first and second pixel dimension).

7.19.1 Detailed Description

7.19.2 Function Documentation

7.19.2.1 HxImageRep L_HXIMAGEREP HxAtan2 (HxImageRep *im*)

Arc tangent (use first and second pixel dimension).

The function computes the arc tangent (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoAtan2** (p. 1212). The image functor instantiator : **HxInstantiatorAtan2** (p. 871).

```
15 {
16     HxString fname("HxAtan2");
17
18     if (im.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23
24     if (im.signature().pixelDimensionality() != 2)
25     {
26         HxGlobalError::instance()->reportError(fname, "atan2 is only defined on two dimensional pixel values", HxGlob
27         return HxImageRep();
28     }
29
30     return im.unaryPixOp("atan2");
31 }
```

7.20 HxBlob2dList.h File Reference

```
#include "HxStd.h"
#include <vector>
```

Compounds

- class **HxBlob2dList**
  A list of HxBlob2dPtr's, that is pointers to **HxBlob2d** (p. 402)*'s.*

Typedefs

- typedef HxBlob2dList::iterator **HxBlob2dListIter**
  Iterator for **HxBlob2dList** (p. 410).
- typedef HxBlob2dList::const_iterator **HxBlob2dListConstIter**
  Const iterator for **HxBlob2dList** (p. 410).
- typedef **HxBlob2dList::back_insert_iterator HxBlob2dListBackInserter**
  Back inserter for **HxPointList** (p. 1073).

7.20.1 Detailed Description

7.20.2 Typedef Documentation

7.20.2.1 typedef HxBlob2dList::iterator HxBlob2dListIter

Iterator for **HxBlob2dList** (p. 410).

7.20.2.2 typedef HxBlob2dList::const_iterator HxBlob2dListConstIter

Const iterator for **HxBlob2dList** (p. 410).

7.20.2.3 typedef HxBlob2dList::back_insert_iterator HxBlob2dListBackInserter

Back inserter for **HxPointList** (p. 1073).

7.21 HxByte.h File Reference

```
#include "HxClassName.h"
```

Compounds

- struct **HxClassName**< **HxByte** >

Typedefs

- typedef unsigned char **HxByte**
  Type definition for byte.

7.21.1 Detailed Description

7.21.2 Typedef Documentation

7.21.2.1 typedef unsigned char HxByte

Type definition for byte.

7.22 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.

7.22.1 Detailed Description

7.22.2 Function Documentation

7.22.2.1 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     HxString fname("HxCannyEdgeMap");
18
19     if (img.isNull())
20     {
21         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
22         return HxImageRep();
23     }
24
25     if (sigma <= 0.0)
26     {
27         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of sigma", HxGlobalError::HX_GE_INVA
28         return HxImageRep();
29     }
30
31     if (img.signature().imageDimensionality() != 2)
32     {
33         HxGlobalError::instance()->reportError(fname, "only defined for 2D images", HxGlobalError::HX_GE_INVALID);
34         return HxImageRep();
35     }
36     if (img.signature().pixelDimensionality() != 1)
37     {
38         HxGlobalError::instance()->reportError(fname, "only defined for scalar pixel types", HxGlobalError::HX_GE_INV
39         return HxImageRep();
40     }
41
42     int minSize = HxImageMinSize(img);
43
44     HxImageRep gauss0 = HxMakeGaussian1d(sigma, 0, 4.0, minSize);
45     HxImageRep gauss1 = HxMakeGaussian1d(sigma, 1, 4.0, minSize);
46
47     HxImageRep Ix = img.genConv2dSep(gauss1, gauss0, "mul", "addAssign",
48                                      HxImageRep::ARITH_PREC);
49     HxImageRep Iy = img.genConv2dSep(gauss0, gauss1, "mul", "addAssign",
50                                      HxImageRep::ARITH_PREC);
51
52     return HxMakeFrom2Images(Ix, Iy);
53 }
```

7.23 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.

7.23.1 Detailed Description

7.23.2 Function Documentation

7.23.2.1 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     HxString fname("HxCannyThreshold");
19
20     if (img.isNull())
21     {
22         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
23         return HxImageRep();
24     }
25
26     if (sigma <= 0.0)
27     {
28         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of sigma", HxGlobalError::HX_GE_INVA
29         return HxImageRep();
30     }
31
32     if (img.signature().imageDimensionality() != 2)
33     {
34         HxGlobalError::instance()->reportError(fname, "only defined for 2D images", HxGlobalError::HX_GE_INVALID);
35         return HxImageRep();
36     }
37     if (img.signature().pixelDimensionality() != 1)
38     {
39         HxGlobalError::instance()->reportError(fname, "only defined for scalar pixel types", HxGlobalError::HX_GE_INV
40         return HxImageRep();
41     }
42
43     HxImageRep edges = HxCannyEdgeMap(img, sigma);
44     HxImageRep nonmax = HxNonMaxSuppressionGradDir(edges);
45     return HxThreshold(HxNorm2Sqr(nonmax), HxValue(level * level));
46 }
```

7.24 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).

7.24.1 Detailed Description

7.24.2 Function Documentation

7.24.2.1 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     HxString fname("HxCannyThresholdAlt");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22
23     if (sigma <= 0.0)
24     {
25         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of sigma", HxGlobalError::HX_GE_INVA
26         return HxImageRep();
27     }
28
29     if (img.signature().imageDimensionality() != 2)
30     {
31         HxGlobalError::instance()->reportError(fname, "only defined for 2D images", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (img.signature().pixelDimensionality() != 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "only defined for scalar pixel types", HxGlobalError::HX_GE_INV
37         return HxImageRep();
38     }
39
40     HxImageRep edges = HxCannyEdgeMap(img, sigma);
41     HxTagList tags;
42     HxAddTag(tags, "level", HxValue(level));
43     return edges.neighbourhoodOp("isMaxGradDir", tags);
44 }
```

7.25 HxCannyThresholdRec.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxCannyThresholdRec** (**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 (implementation uses recursive filters for edge computation).

7.25.1 Detailed Description

7.25.2 Function Documentation

7.25.2.1 HxImageRep L_HXIMAGEREP HxCannyThresholdRec (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 (implementation uses recursive filters for edge computation).

```
15 {
16     HxImageRep Ix = HxRecGauss(img, sigma, sigma, 1, 0, 3);
17     HxImageRep Iy = HxRecGauss(img, sigma, sigma, 0, 1, 3);
18     HxImageRep edges = HxMakeFrom2Images(Ix, Iy);
19
20     HxTagList tags;
21     HxAddTag(tags, "level", HxValue(level));
22     return edges.neighbourhoodOp("isMaxGradDir", tags);
23 }
```

7.26 HxCeil.h File Reference

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

Functions

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

  Ceiling.

7.26.1 Detailed Description

7.26.2 Function Documentation

7.26.2.1 HxImageRep L_HXIMAGEREP HxCeil (HxImageRep *im*)

Ceiling.

The function computes the ceiling (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoCeil** (p. 1214). The image functor instantiator : **HxInstantiatorCeil** (p. 871).

```
13 {
14     HxString fname("HxCeil");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("ceil");
23 }
```

7.27 HxClosing.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxClosing** (**HxImageRep** im, **HxSF** sf)

7.27.1 Detailed Description

7.28 HxClosingByReconstruction.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxClosingByReconstruction** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)

  function y=mmcloserec_equ(f, bdil, bc) y = mmsuprec(mmdil(f,bdil),f,bc);.

7.28.1 Detailed Description

7.28.2 Function Documentation

7.28.2.1 HxImageRep L_HXIMAGEREP HxClosingByReconstruction (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

function y=mmcloserec_equ(f, bdil, bc) y = mmsuprec(mmdil(f,bdil),f,bc);.

```
24 {
25     return HxSupremumReconstruction(HxDilation(im,sf1), im, sf2);
26 }
```

7.29 HxClosingByReconstructionTopHat.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxClosingByReconstructionTopHat** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)

  function y=mmcloserecth_equ(f, bdil, bc) y = mmsubm(mmcloserec(f, bdil, bc),f);.

7.29.1 Detailed Description

7.29.2 Function Documentation

7.29.2.1 HxImageRep L_HXIMAGEREP HxClosingByReconstructionTopHat (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

function y=mmcloserecth_equ(f, bdil, bc) y = mmsubm(mmcloserec(f, bdil, bc),f);.

```
26 {
27     return HxSubSat(HxClosingByReconstruction(im,sf1,sf2), im);
28 }
```

7.30 HxClosingTopHat.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxClosingTopHat** (**HxImageRep** im, **HxSF** sf)

  res = closing(im,sf) - im ? (SDC morph) or res = im - closing(im,sf) ? (Rein).

7.30.1 Detailed Description

7.30.2 Function Documentation

7.30.2.1 HxImageRep L_HXIMAGEREP HxClosingTopHat (HxImageRep *im*, HxSF *sf*)

res = closing(im,sf) - im ? (SDC morph) or res = im - closing(im,sf) ? (Rein).

```
16 {
17     return HxSub(HxClosing(im,sf), im);
18
19 }
```

7.31 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. 1301) *to an ARGB integer representation.*
- L_HXBASIS int **HxColRGB2int** (const **HxVec3Int** &v)
  Convert RGB **HxVec3Int** (p. 1321) *to an ARGB integer representation.*

7.31.1 Detailed Description

7.31.2 Function Documentation

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

Conversion from RGB to CMY.

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

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

Conversion from CMY to RGB.

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

7.31.2.3 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 }
```

7.31.2.4 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 }
```

7.31.2.5 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 }
```

7.31.2.6 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 }
```

7.31.2.7 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 }
```

7.31.2.8 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 }
```

7.31.2.9 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 }
```

7.31.2.10 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 }
```

7.31.2.11 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 }
```

7.31.2.12 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 }
```

7.31.2.13 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 }
```

7.31.2.14 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 }
```

7.31.2.15 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 }
```

7.31.2.16 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 }
```

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

Convert RGB **HxVec3Double** (p. 1301) to an ARGB integer representation.

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

Convert RGB **HxVec3Int** (p. 1321) 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 }
```

7.32 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)

7.32.1 Detailed Description

7.32.2 Enumeration Type Documentation

7.32.2.1 enum HxColorModel

Supported color spaces.

See also : **Color operations on pixel values** (p. 6).

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

7.33 HxColorSpace.h File Reference

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

Functions

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

  Color space conversion.

7.33.1 Detailed Description

7.33.2 Function Documentation

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

Color space conversion.

The function transforms the color model (see **Color operations on pixel values** (p. 6)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoColSpace** (p. 1215). The image functor instantiator : **HxInstantiatorColSpace** (p. 872).

```
14 {
15     HxString fname("HxColorSpace");
16
17     if (im.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (im.signature().pixelDimensionality() != 3)
23     {
24         HxGlobalError::instance()->reportError(fname, "ColorSpace conversions are only valid for Vec3 images", HxGlob
25         return HxImageRep();
26     }
27
28     HxTagList tags;
29     HxAddTag<HxColorModel>(tags, "fromColorSpace", fromColorSpace);
30     HxAddTag<HxColorModel>(tags, "toColorSpace", toColorSpace);
31     return im.unaryPixOp("colorSpace", tags);
32 }
```

7.34 HxComplement.h File Reference

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

Functions

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

  One's complement.

7.34.1 Detailed Description

7.34.2 Function Documentation

7.34.2.1 HxImageRep L_HXIMAGEREP HxComplement (HxImageRep *im*)

One's complement.

The function computes the one's complement (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoComplement** (p. 1217). The image functor instantiator : **HxInstantiatorComplement** (p. 872).

```
13 {
14     HxString fname("HxComplement");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.signature().pixelType() != INT_VALUE)
22     {
23         HxGlobalError::instance()->reportError(fname, "this function is only valid on integer values", HxGlobalError
24         return HxImageRep();
```

```
25     }
26
27     return im.unaryPixOp("complement");
28 }
```

7.35 HxConditionalDilation.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConditionalDilation** (**HxImageRep** im, **HxImageRep** mask, **HxSF** sf, int nrIter=1)

  function y=mmcdil_equ(f,g,b,n) y = mmintersec(f,g); for i=1:n dil = mmdil(y,b); y = mmintersec(dil,g); end.

7.35.1 Detailed Description

7.35.2 Function Documentation

7.35.2.1 HxImageRep L_HXIMAGEREP HxConditionalDilation (HxImageRep *im*, HxImageRep *mask*, HxSF *sf*, int *nrIter* = 1)

function y=mmcdil_equ(f,g,b,n) y = mmintersec(f,g); for i=1:n dil = mmdil(y,b); y = mmintersec(dil,g); end.

default value=1. function y=mmcdil_equ(f,g,b,n) y = mmintersec(f,g); for i=1:n dil = mmdil(y,b); y = mmintersec(dil,g); end

NOTE: mmunion takes the maximum of the two images

```
39 {
40
41     HxImageRep res, ero;
42
43     res = ::HxMin(im, mask);
44 //  res=im.binaryPixOp(mask, "min");
45
49
50     HxImageRep tmp1, tmp2;
51     int ncheck=10;
52     tmp1 = res;
53
54     for(int i=0; i< nrIter; i++)
55     {
56         ero = HxDilation(res, sf);
57         res = ::HxMin(ero, mask);
58
59         if( i% ncheck ==1)
60         {
61             tmp2 = res;
62             if( HxPixSum(::HxEqual(tmp1, tmp2)).HxScalarIntValue().x() == res.numberOfPixels() )
63                 break;
64             else
65                 tmp1 = tmp2;
```

```
66         }
67
68     }
69
70
71     return res;
72 }
```

7.36 HxConditionalErosion.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxConditionalErosion** (**HxImageRep** im, **HxImageRep** mask, **HxSF** sf, int nrIter=1)

  /input im /input mask /input sf /input nrIter number of iterations.

7.36.1 Detailed Description

7.36.2 Function Documentation

7.36.2.1 HxImageRep L_HXIMAGEREP HxConditionalErosion (HxImageRep *im*, HxImageRep *mask*, HxSF *sf*, int *nrIter* = 1)

/input im /input mask /input sf /input nrIter number of iterations.

default value=1. function y=mmcero_equ(f,g,b,n)

y = mmunion(f,g);

for i=1:n

ero = mmero(y,b);

y = mmunion(ero,g);

end

NOTE: mmunion takes the maximum of the two images

```
24 {
25
26     HxImageRep res, ero;
27
28     res = ::HxMax(im, mask);
29
30     HxImageRep tmp1, tmp2;
31     int ncheck=10;
32     tmp1 = res;
33
34     for(int i=0; i< nrIter; i++)
35     {
36         ero = HxErosion(res, sf);
37         res = ::HxMax(ero, mask);
38
```

```
39         if( i% ncheck ==0)
40         {
41             tmp2 = res;
42             if( HxPixSum(HxEqual(tmp1, tmp2)).HxScalarIntValue() == res.numberOfPixels() )
43                 break;
44             else
45                 tmp1 = tmp2;
46         }
47     }
48     return res;
49 }
```

7.37 HxConjugate.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxConjugate** (**HxImageRep** im)

  Complex conjugate.

7.37.1 Detailed Description

7.37.2 Function Documentation

7.37.2.1 HxImageRep L_HXIMAGEREP HxConjugate (HxImageRep *im*)

Complex conjugate.

The function computes the complex conjugate (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoConjugate** (p. 1218). The image functor instantiator : **HxInstantiatorConjugate** (p. 873).

```
13 {
14     HxString fname("HxConjugate");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     if (im.signature().pixelType() != COMPLEX_VALUE)
23     {
24         HxGlobalError::instance()->reportError(fname, "Operation is only valid on complex images", HxGlobalError::HX_
25         return HxImageRep();
26     }
27
28     return im.unaryPixOp("conjugate");
29 }
```

7.38 HxContrastStretch.h File Reference

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

Functions

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

7.38.1 Detailed Description

7.38.2 Function Documentation

7.38.2.1 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     HxString fname("HxContrastStretch");
21
22     if (img.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27
28     HxScalarDouble min = HxPixMin(img);
29     HxScalarDouble max = HxPixMax(img);
30     if (min <= max)
31     {
32         HxGlobalError::instance()->reportError(fname, img.name(), "maximum is less than or equal to minimum is not al
33         return HxImageRep();
34     }
35     if (mFactor < (max.x() - min.x()))
36     {
37         HxGlobalError::instance()->reportError(fname, img.name(), "mFactor is less than original range", HxGlobalErro
38         return HxImageRep();
39     }
40
41     return HxMulVal(HxSubVal(img, min), mFactor / (max - min));
42 }
```

7.39 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.

7.39.1 Detailed Description

7.39.2 Function Documentation

7.39.2.1 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     HxString fname("HxConvGauss2d");
20 
21     HxImageRep gaussx, gaussy;
22 
23     if (img.isNull())
24     {
25         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28     if (sigmax <= 0.0)
29     {
30         HxGlobalError::instance()->reportError(fname, img.name(), "sigmax is equal to or less than zero", HxGlobalEr
31         return HxImageRep();
32     }
33     if (sigmay <= 0.0)
34     {
35         HxGlobalError::instance()->reportError(fname, img.name(), "sigmay is equal to or less than zero", HxGlobalEr
36         return HxImageRep();
37     }
38     if (orderDerivx < 0)
39     {
40         HxGlobalError::instance()->reportError(fname, img.name(), "orderDerivx is less than 0", HxGlobalError::HX_GE_
41         return HxImageRep();
42     }
43     if (orderDerivy < 0)
44     {
45         HxGlobalError::instance()->reportError(fname, img.name(), "orderDerivy is less than 0", HxGlobalError::HX_GE_
46         return HxImageRep();
47     }
48     if (truncationx < 0)
49     {
50         HxGlobalError::instance()->reportError(fname, img.name(), "truncationx is less than 0", HxGlobalError::HX_GE_
51         return HxImageRep();
52     }
53     if (truncationy < 0)
54     {
55         HxGlobalError::instance()->reportError(fname, img.name(), "truncationy is less than 0", HxGlobalError::HX_GE_
56         return HxImageRep();
57     }
```

```
58
59     if (img.dimensionality() != 2)
60     {
61         HxGlobalError::instance()->reportError(fname, "only defined for 2D images", HxGlobalError::HX_GE_INVALID);
62         return HxImageRep();
63     }
64
65     gaussx = HxMakeGaussian1d(sigmax, orderDerivx, truncationx,
66         img.dimensionSize(1));
67     gaussy = HxMakeGaussian1d(sigmay, orderDerivy, truncationy,
68         img.dimensionSize(2));
69
70     return img.genConv2dSep(
71         gaussx, gaussy, "mul", "addAssign", HxImageRep::ARITH_PREC);
72 }
```

7.40 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.

7.40.1 Detailed Description

7.40.2 Function Documentation

7.40.2.1 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: img.genConv3dSep(gaussx, gaussy, gaussz, ”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     HxString fname("HxConvGauss3d");
21
22     HxImageRep gaussx, gaussy, gaussz;
23
24     if (img.isNull())
25     {
26         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
```

```
27          return HxImageRep();
28      }
29      if (sigmax <= 0.0)
30      {
31          HxGlobalError::instance()->reportError(fname, img.name(), "sigmax is equal to or less than zero", HxGlobalErr
32          return HxImageRep();
33      }
34      if (sigmay <= 0.0)
35      {
36          HxGlobalError::instance()->reportError(fname, img.name(), "sigmay is equal to or less than zero", HxGlobalErr
37          return HxImageRep();
38      }
39      if (sigmaz <= 0.0)
40      {
41          HxGlobalError::instance()->reportError(fname, img.name(), "sigmaz is equal to or less than zero", HxGlobalErr
42          return HxImageRep();
43      }
44      if (orderDerivx < 0)
45      {
46          HxGlobalError::instance()->reportError(fname, img.name(), "orderDerivx is less than 0", HxGlobalError::HX_GE_
47          return HxImageRep();
48      }
49      if (orderDerivy < 0)
50      {
51          HxGlobalError::instance()->reportError(fname, img.name(), "orderDerivy is less than 0", HxGlobalError::HX_GE_
52          return HxImageRep();
53      }
54      if (orderDerivz < 0)
55      {
56          HxGlobalError::instance()->reportError(fname, img.name(), "orderDerivz is less than 0", HxGlobalError::HX_GE_
57          return HxImageRep();
58      }
59
60      if (truncationx < 0)
61      {
62          HxGlobalError::instance()->reportError(fname, img.name(), "truncationx is less than 0", HxGlobalError::HX_GE_
63          return HxImageRep();
64      }
65      if (truncationy < 0)
66      {
67          HxGlobalError::instance()->reportError(fname, img.name(), "truncationy is less than 0", HxGlobalError::HX_GE_
68          return HxImageRep();
69      }
70      if (truncationz < 0)
71      {
72          HxGlobalError::instance()->reportError(fname, img.name(), "truncationz is less than 0", HxGlobalError::HX_GE_
73          return HxImageRep();
74      }
75
76      if (img.signature().imageDimensionality() != 3)
77      {
78          HxGlobalError::instance()->reportError(fname, "only defined for 3D images", HxGlobalError::HX_GE_INVALID);
79          return HxImageRep();
80      }
81
82      gaussx = HxMakeGaussian1d(sigmax, orderDerivx, truncationx,
83          img.dimensionSize(1));
84      gaussy = HxMakeGaussian1d(sigmay, orderDerivy, truncationy,
85          img.dimensionSize(2));
86      gaussz = HxMakeGaussian1d(sigmaz, orderDerivz, truncationz,
87          img.dimensionSize(3));
88
89      return img.genConv3dSep(gaussx, gaussy, gaussz,
90                              "mul", "addAssign", HxImageRep::ARITH_PREC);
91  }
```

7.41 HxConvKernelSeparated.h File Reference

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

Functions

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

  Convolution with separable kernel.

7.41.1 Detailed Description

7.41.2 Function Documentation

7.41.2.1 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. 8)). The same kernel is applied in each dimension.

Implementation specifics : The image functor instantiator for 2D images : HxInstMulAddAss2dK1d, and for 3D images : HxInstMulAddAss3dK1d.

```
15 {
16     HxString fname("HxConvGauss2d");
17
18     if (img.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if (kernel.isNull())
24     {
25         HxGlobalError::instance()->reportError(fname, kernel.name(), "null kernel", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28     if (kernel.signature().pixelDimensionality() != 1)
29     {
30         HxGlobalError::instance()->reportError(fname, "only defined for scalar kernel pixel types", HxGlobalError::HX
31         return HxImageRep();
32     }
33     if (kernel.dimensionSize(2) != 1)
34     {
35         HxGlobalError::instance()->reportError(fname, "N x 1 kernel image required", HxGlobalError::HX_GE_INVALID);
36         return HxImageRep();
37     }
38     if (kernel.dimensionSize(1) > img.dimensionSize(1))
39     {
40         HxGlobalError::instance()->reportError(fname, "kernel size larger than image x-dimension size", HxGlobalError
41         return HxImageRep();
42     }
43     if ((img.signature().imageDimensionality() > 2) &&
44         (kernel.dimensionSize(1) > img.dimensionSize(2)))
45     {
46         HxGlobalError::instance()->reportError(fname, "kernel size larger than image y-dimension size", HxGlobalError
```

```
47         return HxImageRep();
48     }
49     if ((img.signature().imageDimensionality() > 2) &&
50         (kernel.dimensionSize(1) > img.dimensionSize(3)))
51     {
52         HxGlobalError::instance()->reportError(fname, "kernel size larger than image z-dimension size", HxGlobalError
53         return HxImageRep();
54     }
55
56     return img.genConvSeparated(kernel, "mul", "addAssign", resPrec);
57 }
```

7.42 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)

7.42.1 Detailed Description

7.42.2 Function Documentation

7.42.2.1 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. 8)). KernelX is applied in the first dimension, kernelY in the second dimension.

Implementation specifics : The image functor instantiator : HxInstMulAddAss2dK1d.

7.43 HxConvolution.h File Reference

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

Functions

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

  Convolution.

7.43.1 Detailed Description

7.43.2 Function Documentation

7.43.2.1 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. 8)).

Implementation specifics : The image functor instantiator for 2D images: HxInstMulAddAss2d, and for 3D images : HxInstMulAddAss3d.

```
15 {
16     HxString fname("HxConvolution");
17 
18     if (img.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if (kernel.isNull())
24     {
25         HxGlobalError::instance()->reportError(fname, kernel.name(), "null kernel", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28     if (img.dimensionality() != kernel.dimensionality())
29     {
30         HxGlobalError::instance()->reportError(fname, "kernel and image dimensionality do not match", HxGlobalError:
31         return HxImageRep();
32     }
33     if (kernel.pixelDimensionality() != img.pixelDimensionality())
34     {
35         HxGlobalError::instance()->reportError(fname, "kernel and image pixel dimensionality do not match", HxGlobalE
36         return HxImageRep();
37     }
38 
39     return img.generalizedConvolution(
40             kernel, "mul", "addAssign", resPrec);
41 }
```

7.44 HxCos.h File Reference

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

Functions

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

  Cosine.

7.44.1 Detailed Description

7.44.2 Function Documentation

7.44.2.1 HxImageRep L_HXIMAGEREP HxCos (HxImageRep *im*)

Cosine.

The function computes the cosine (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoCos** (p. 1220). The image functor instantiator : **HxInstantiatorCos** (p. 874).

```
13 {
14     HxString fname("HxCos");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("cos");
23 }
```

7.45 HxCosh.h File Reference

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

Functions

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

  Hyperbolic cosine.

7.45.1 Detailed Description

7.45.2 Function Documentation

7.45.2.1 HxImageRep L_HXIMAGEREP HxCosh (HxImageRep *im*)

Hyperbolic cosine.

The function computes the hyperbolic cosine (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoCosh** (p. 1221). The image functor instantiator : **HxInstantiatorCosh** (p. 874).

```
13 {
14     HxString fname("HxCosh");
15
16     if (im.isNull())
```

```
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("cosh");
23 }
```

7.46 HxCross.h File Reference

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

Functions

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

  Cross product.

7.46.1 Detailed Description

7.46.2 Function Documentation

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

Cross product.

The function performs cross product (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoCross** (p. 425). The image functor instantiator : **HxInstantiatorCross** (p. 875).

```
13 {
14     HxString fname("HxCross");
15
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32
33     if ((im1.pixelDimensionality() != 3) || (im2.pixelDimensionality() != 3))
34     {
35         HxGlobalError::instance()->reportError(fname, "Operation is only valid for vec3 images", HxGlobalError::HX_GI
36         return HxImageRep();
```

```
37     }
38
39     if (im1.sizes().x() != im2.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im1.sizes().y() != im2.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (im1.dimensionality() > 2)
50     {
51         if (im1.sizes().z() != im2.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
54             return HxImageRep();
55         }
56     }
57
58     return im1.binaryPixOp(im2, "cross");
59 }
```

7.47 HxCrossVal.h File Reference

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

Functions

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

  Cross product.

7.47.1 Detailed Description

7.47.2 Function Documentation

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

Cross product.

The function performs cross product (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoCross** (p. 425). The image functor instantiator : **HxInstantiatorCrossV** (p. 875).

```
13 {
14     HxString fname("HxCrossVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
```

```
21
22     if (im.pixelDimensionality() != 3)
23     {
24         HxGlobalError::instance()->reportError(fname, "Operation is only valid for vec3 image", HxGlobalError::HX_GE_
25         return HxImageRep();
26     }
27
28
29     if (im.signature().pixelType() != INT_VALUE)
30     {
31         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
32         return HxImageRep();
33     }
34
35     if ((val.tag() != HxValue::V3I) && (val.tag() != HxValue::V3D))
36     {
37         HxGlobalError::instance()->reportError(fname, "operation is only valid for vec3 values", HxGlobalError::HX_GE
38         return HxImageRep();
39     }
40
41     return im.binaryPixOp(val, "cross");
42 }
```

7.48 HxDefuz.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxDefuz** (**HxImageRep** im, int windowSzX=5, int windowSzY=5, double thr=0.5)

  (from ScilImage Help) defuz() performs a sharpening operation on the grey value image ”in” and stores the result in the grey value image ”out”.

7.48.1 Detailed Description

7.48.2 Function Documentation

7.48.2.1 HxImageRep L_HXIMAGEREP HxDefuz (HxImageRep *im*, int *windowSzX* = 5, int *windowSzY* = 5, double *thr* = 0.5)

(from ScilImage Help) defuz() performs a sharpening operation on the grey value image ”in” and stores the result in the grey value image ”out”.

The image is scanned with a moving window with sizes ”filt_x” and ”filt_y”. For each position, both minimum and maximum value are determined. The values of the minimum, maximum and center pixel value are denoted by MIN, MAX and C resp. The value of the last parameter ”thr” is denoted by THR. The new value of the center pixel is calculated as follows:

if(C $<$ MIN + THR $*$ (MAX - MIN)) C = MIN; else C = MAX;

The last parameter THR indicates a bias towards either the local minimum or the local maximum value. In case THR == 0.5 there is no bias towards a direction. This means that if THR == 0.5 the center value is substituted by either min or max depending on whichever of the two is closest in value. If THR has value

0 the result is identical to a local maximum operation and if THR has value 1 the result is that of a local minimum filter. For non-linear sharpening the value of 0.5 is recommended for THR.

The bernsen_threshold() operation is comparable to defuz() without bias. However, instead of replacing the value of the center pixel by the local minimum or maximum, this pixel is assigned 0 or 1. The result of this operation will become very noisy in areas where there is no distinct difference between minimum and maximum. Therefore, with the last parameter "max_diff" the user can indicate the minimum required difference between the local minimum and maximum. If the difference is less, the center pixel is assigned 1 by default.

```
12 {
13     HxTagList tags;
14     HxAddTag(tags, "windowSzX", windowSzX);
15     HxAddTag(tags, "windowSzY", windowSzY);
16     HxAddTag(tags, "thr", thr);
17
18     return im.neighbourhoodOp("defuz", tags);
19
20 }
```

7.49 HxDilation.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDilation** (**HxImageRep** im, **HxSF** sf)

7.49.1 Detailed Description

7.50 HxDisplayOF.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxDisplayOF** (**HxImageRep** in, int scale_x, int scale_y, double mul_x, double mul_y, int pixelsize)

  Display of optical flow.

7.50.1 Detailed Description

7.50.2 Function Documentation

7.50.2.1 HxImageRep L_HXIMAGEREP HxDisplayOF (HxImageRep *in*, int *scale_x*, int *scale_y*, double *mul_x*, double *mul_y*, int *pixelsize*)

Display of optical flow.

```
252 {
253     HxSizes sz= im.sizes();
254 
255     HxVec3Byte bkcolor(200,200,200);
256 
257     HxImageRep out_f = HxMakeFromValue(HXIMAGESIG2DVEC3BYTE,sz,(200,200,200));
258     HxVec3Double color(0,0,255);
259 
260      int     fx = sz.x();
261      int     fy = sz.y();
262      int ox = fx*pixelsize;
263      int oy = fy*pixelsize;
264 
265      for(int y = 0; y < fy; y += scale_y ) {
266          for(int x = 0; x < fx; x += scale_x ) {
267              long npix = y*fx + x;
268              float vu = -(im.getAt(x,y).HxVec2DoubleValue().x());
269              float vv = -(im.getAt(x,y).HxVec2DoubleValue().y());
270 
271              int qx = mul_x*vu;
272              int qy = mul_y*vv;
273 
274              if(qx!=0 || qy!=0)
275              {
276 
277              int xn = pixelsize*x;
278              int yn = pixelsize*y;
279 
280              int dx = xn + qx; dx = max(dx,0); dx = min((ox-1),dx);
281              int dy = yn + qy; dy = max(dy,0); dy = min((oy-1),dy);
282              HxDrawLine(out_f, xn, yn, dx, dy, color);
283 
284              int tx = dx - ((qx - qy)>>3);  tx = max(tx,0); tx = min((ox-1),tx);
285              int ty = dy - ((qy + qx)>>3);  ty = max(ty,0); ty = min((oy-1),ty);
286              HxDrawLine(out_f, dx, dy, tx, ty, HxVec3Double(255,0,0));
287 
288              tx = dx - ((qx + qy)>>3);  tx = max(tx,0); tx = min((ox-1),tx);
289              ty = dy - ((qy - qx)>>3);  ty = max(ty,0); ty = min((oy-1),ty);
290              HxDrawLine(out_f, dx, dy, tx, ty, HxVec3Double(255,0,0));
291              }
292          }
293      }
294     return out_f;
295 }
```

7.51 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).

7.51.1 Detailed Description

7.51.2 Function Documentation

7.51.2.1 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.

```
89 {
90     HxString fname("HxDistanceTransform");
91
92     if (img.pixelDimensionality() != 1)
93     {
94         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_UI
95         return HxImageRep();
96     }
97
98     HxImageRep input = BinMap(HxImageAsDouble(img), inf, 0);
99
100     double filterData[] = {
101         inf,    sqrt_5, inf,    sqrt_5, inf,
102         sqrt_5, sqrt_2, 1,      sqrt_2, sqrt_5,
103         inf,    1,      0,      1,      inf,
104         sqrt_5, sqrt_2, 1,      sqrt_2, sqrt_5,
105         inf,    sqrt_5, inf,    sqrt_5, inf
106     };
107     HxImageRep kernel = HxMakeFromDoubleData(1, 2, HxSizes(5, 5, 1), filterData);
108
109     return input.recGenConv(kernel, "add", "minAssign");
110 }
```

7.52 HxDistanceTransformMM.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxDistanceTransformMM** (**HxImageRep** im, **HxSF** sf)
  /input f: image /input Bc: SF.

7.52.1 Detailed Description

7.52.2 Function Documentation

7.52.2.1 HxImageRep L_HXIMAGEREP HxDistanceTransformMM (HxImageRep *im*, HxSF *sf*)

/input f: image /input Bc: SF.

this is defined for BW image only im mmorph algorithm

z = 0; g = f; ero = f;

while (ero != z) ero = HxErosion(ero,Bc); g = mmaddm(g,mmgray(ero,'uint8',1)); end

NOTE: mmadd is addition with saturation!

```
48 {
49     HxImageRep ero, res;
50
51     if( HxPixMax(im).HxScalarIntValue() > 1 )
52     {
53 //      std::cout<<im.signature() << std::endl;
54 //      std::cout<< HxPixMax(im) << std::endl;
55         std::cout<< std::endl<<"HxDistanceTransformMM works only on binary images!" << std::endl;
56
57         return res; //return null image
58     }
59
60
61
62     res = im;
63     ero = im;
64
65     HxValue zero(0);
66
67     int i=0;
68 //  while( HxScalarInt(HxPixMax(ero)) != 0 )
69 //  while( i < 50 )
70     while( HxPixSum(HxEqualVal(ero, 0)).HxScalarIntValue() != ero.numberOfPixels() )
71     {
72         i++;
73         ero = HxErosion(ero, sf);
74         res = HxAddSat( res, ero);
75     }
76
77     return res;
78 }
```

7.53 HxDiv.h File Reference

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

Functions

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

7.53.1 Detailed Description

7.53.2 Function Documentation

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

Division.

The function performs division (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoDiv** (p. 426). The image functor instantiator : **HxInstantiatorDiv** (p. 876).

```
13 {
14     HxString fname("HxDiv");
15
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32     if ((im1.pixelDimensionality() != im2.pixelDimensionality()) &&
33         (im2.pixelDimensionality() != 1))
34     {
35         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
36         return HxImageRep();
37     }
38
39     if (im1.sizes().x() != im2.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im1.sizes().y() != im2.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (im1.dimensionality() > 2)
50     {
51         if (im1.sizes().z() != im2.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGE
54             return HxImageRep();
55         }
56     }
57
58     return im1.binaryPixOp(im2, "div");
59 }
```

7.54 HxDivVal.h File Reference

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

Functions

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

7.54.1 Detailed Description

7.54.2 Function Documentation

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

Division.

The function performs division (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoDiv** (p. 426). The image functor instantiator : **HxInstantiatorDivV** (p. 877).

```
13 {
14     HxString fname("HxDivVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if ((valdim != 1) && (im.signature().pixelDimensionality() != valdim))
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     if (val.tag() == HxValue::SI)
43     {
44         if (((HxScalarInt) val) == HxScalarInt(0))
45         {
46             HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
47             return HxImageRep();
48         }
49     }
50     else if (val.tag() == HxValue::SD)
51     {
52         if (((HxScalarDouble) val) == HxScalarDouble(0.0))
53         {
54             HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
55             return HxImageRep();
56         }
57     }
58     else if (val.tag() == HxValue::V2I)
59     {
60         if (((HxVec2Int) val).x() == 0)
61         {
```

```
62              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
63              return HxImageRep();
64          }
65          if (((HxVec2Int) val).y() == 0)
66          {
67              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
68              return HxImageRep();
69          }
70      }
71      else if ((val.tag() == HxValue::V2D) || (val.tag() == HxValue::CPL))
72      {
73          if (((HxVec2Double) val).x() == 0.0)
74          {
75              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
76              return HxImageRep();
77          }
78          if (((HxVec2Double) val).y() == 0.0)
79          {
80              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
81              return HxImageRep();
82          }
83
84      }
85      else if (val.tag() == HxValue::V3I)
86      {
87          if (((HxVec3Int) val).x() == 0)
88          {
89              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
90              return HxImageRep();
91          }
92          if (((HxVec3Int) val).y() == 0)
93          {
94              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
95              return HxImageRep();
96          }
97          if (((HxVec3Int) val).z() == 0)
98          {
99              HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
100             return HxImageRep();
101         }
102     }
103     else if (val.tag() == HxValue::V3D)
104     {
105         if (((HxVec3Double) val).x() == 0.0)
106         {
107             HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
108             return HxImageRep();
109         }
110         if (((HxVec3Double) val).y() == 0.0)
111         {
112             HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
113             return HxImageRep();
114         }
115         if (((HxVec3Double) val).z() == 0.0)
116         {
117             HxGlobalError::instance()->reportError(fname, "division by zero", HxGlobalError::HX_GE_INVALID);
118             return HxImageRep();
119         }
120     }
121
122     return im.binaryPixOp(val, "div");
123 }
```

7.55 HxDot.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxDot** (**HxImageRep** im1, **HxImageRep** im2)
  Dot product .

7.55.1 Detailed Description

7.55.2 Function Documentation

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

Dot product .

The function performs dot product (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoDot** (p. 428). The image functor instantiator : **HxInstantiatorDot** (p. 877).

```
13 {
14     HxString fname("HxDot");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.signature().imageDimensionality() != im2.signature().imageDimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32     if (im1.signature().pixelDimensionality() != im2.signature().pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
35         return HxImageRep();
36     }
37 
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
```

```
47     }
48     if (im1.signature().imageDimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
54         }
55     }
56
57     // in case of byte, unsigned: generate warnings in case of potentially dangerous
58     // situations.
59     // Check if image is byte.
60     if (((im1.signature().pixelType() == INT_VALUE) &&
61          (im1.signature().pixelPrecision() == 8)) ||
62         ((im2.signature().pixelType() == INT_VALUE) &&
63          (im2.signature().pixelPrecision() == 8)))
64     {
65         if ((im1.pixelDimensionality() == 1) && (im1.pixelDimensionality() == 1))
66         {
67             if ((HxPixMax(im1).HxScalarIntValue() +
68                  HxPixMax(im2).HxScalarIntValue()) > HxScalarInt(255))
69             {
70                 HxGlobalError::instance()->reportWarning(fname,
71                     im1.name()+HxString(" ")+im2.name(),
72                     "possible overflow due to byte precision",
73                     HxGlobalError::HX_GW_OVERFLOW);
74             }
75         }
76         else if ((HxPixMax(HxUnaryMax(im1)).HxScalarIntValue() +
77                   HxPixMax(HxUnaryMax(im2)).HxScalarIntValue()) > HxScalarInt(255))
78         {
79                 HxGlobalError::instance()->reportWarning(fname,
80                     im1.name()+HxString(" ")+im2.name(),
81                     "possible overflow due to byte precision",
82                     HxGlobalError::HX_GW_OVERFLOW);
83         }
84     }
85
86     return im1.binaryPixOp(im2, "dot");
87 }
```

7.56 HxDotVal.h File Reference

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

Functions

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

  Dot product .

7.56.1 Detailed Description

7.56.2 Function Documentation

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

Dot product .

The function performs dot product (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoDot** (p. 428). The image functor instantiator : **HxInstantiatorDotV** (p. 878).

```
13 {
14     HxString fname("HxAddVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if (im.signature().pixelDimensionality() != valdim)
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "dot");
43 }
```

7.57 HxEqual.h File Reference

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

Functions

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

  Equal.

7.57.1 Detailed Description

7.57.2 Function Documentation

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

Equal.

The function performs equal (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoEqual** (p. 429). The image functor instantiator : **HxInstantiatorEqual** (p. 879).

```
13 {
14     HxString fname("HxEqual");
15
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if (im1.pixelDimensionality() != im2.pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35         return HxImageRep();
36     }
37
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.dimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
54         }
55     }
56
57     return im1.binaryPixOp(im2, "equal");
58 }
```

7.58 HxEqualVal.h File Reference

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

Functions

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

  Equal.

7.58.1 Detailed Description

7.58.2 Function Documentation

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

Equal.

The function performs equal (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoEqual** (p. 429). The image functor instantiator : **HxInstantiatorEqualV** (p. 879).

```
13 {
14     HxString fname("HxEqualVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if (im.signature().pixelDimensionality() != valdim)
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "equal");
43 }
```

7.59 HxErosion.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxErosion** (**HxImageRep** im, **HxSF** sf)

7.59.1 Detailed Description

7.60 HxExp.h File Reference

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

Functions

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

  Exponent.

7.60.1 Detailed Description

7.60.2 Function Documentation

7.60.2.1 HxImageRep L_HXIMAGEREP HxExp (HxImageRep *im*)

Exponent.

The function computes the exponent (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoExp** (p. 1223). The image functor instantiator : **HxInstantiatorExp** (p. 880).

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

7.61 HxExportByteData.h File Reference

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

Functions

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

  Export image data to array of HxByte.

7.61.1 Detailed Description

7.61.2 Function Documentation

7.61.2.1 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     HxString fname("HxExportByteData");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return;
21     }
22
23     if (data == NULL)
24     {
25         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
26         return;
27     }
28
29     HxTagList tags;
30     HxString exportOp
31         = HxString("exportPix<") + HxString(HxClassName<HxByte>()) + ">";
32     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
33     img.exportOp(exportOp, tags);
34 }
```

7.62 HxExportDoubleData.h File Reference

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

Functions

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

  Export image data to array of double.

7.62.1 Detailed Description

7.62.2 Function Documentation

7.62.2.1 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     HxString fname("HxExportDoubleData");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return;
20     }
21
22     if (data == NULL)
23     {
24         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
25         return;
26     }
27
28     HxTagList tags;
29     HxString exportOp
30         = HxString("exportPix<") + HxString(HxClassName<double>()) + ">";
31     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
32     img.exportOp(exportOp, tags);
33 }
```

7.63 HxExportFloatData.h File Reference

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

Functions

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

  Export image data to array of float.

7.63.1 Detailed Description

7.63.2 Function Documentation

7.63.2.1 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     HxString fname("HxExportFloatData");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return;
20     }
21
22     if (data == NULL)
23     {
24         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
25         return;
```

```
26     }
27
28     HxTagList tags;
29     HxString exportOp
30         = HxString("exportPix<") + HxString(HxClassName<float>()) + ">";
31     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
32     img.exportOp(exportOp, tags);
33 }
```

7.64 HxExportIntData.h File Reference

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

Functions

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

  Export image data to array of int.

7.64.1 Detailed Description

7.64.2 Function Documentation

7.64.2.1 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     HxString fname("HxExportIntData");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return;
20     }
21
22     if (data == NULL)
23     {
24         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
25         return;
26     }
27
28     HxTagList tags;
29     HxString exportOp
30         = HxString("exportPix<") + HxString(HxClassName<int>()) + ">";
31     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
32     img.exportOp(exportOp, tags);
33 }
```

7.65 HxExportMatlabPixels.h File Reference

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

Functions

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

  Export image data to array of int.

7.65.1 Detailed Description

7.65.2 Function Documentation

7.65.2.1 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     HxString fname("HxExportMatlabPixels");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return;
20     }
21
22     if (pixels == NULL)
23     {
24         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
25         return;
26     }
27
28     img.getDoublePixels(pixels);
29 }
```

7.66 HxExportPpmPixels.h File Reference

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

Functions

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

7.66.1 Detailed Description

7.67 HxExportShortData.h File Reference

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

Functions

- void L_HXIMAGEREP **HxExportShortData** (**HxImageRep** img, short *data)
  Export image data to array of short.

7.67.1 Detailed Description

7.67.2 Function Documentation

7.67.2.1 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     HxString fname("HxExportShortData");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return;
20     }
21
22     if (data == NULL)
23     {
24         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
25         return;
26     }
27
28     HxTagList tags;
29     HxString exportOp
30         = HxString("exportPix<") + HxString(HxClassName<short>()) + ">";
31     HxAddTag(tags, "dataPtr", static_cast<void*>(data));
32     img.exportOp(exportOp, tags);
33 }
```

7.68 HxExtend.h File Reference

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

Functions

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

Extension of domain.

7.68.1 Detailed Description

7.68.2 Function Documentation

7.68.2.1 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     HxString fname("HxExtend");
15 
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (background.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, background.name(), "null image", HxGlobalError::HX_GE_INVALID)
24         return HxImageRep();
25     }
26 
27     if (img.dimensionality() != background.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if (img.pixelDimensionality() != background.pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35         return HxImageRep();
36     }
37 
38     if (begin.x() < 0)
39     {
40         HxGlobalError::instance()->reportError(fname, "begin.x less then 0", HxGlobalError::HX_GE_INVALID);
41         return HxImageRep();
42     }
43     if (begin.y() < 0)
44     {
45         HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
46         return HxImageRep();
47     }
48     if (begin.z() < 0)
49     {
50         HxGlobalError::instance()->reportError(fname, "begin.z less then 0", HxGlobalError::HX_GE_INVALID);
51         return HxImageRep();
52     }
53     if ((begin.x() + img.sizes().x()) > background.sizes().x())
54     {
55         HxGlobalError::instance()->reportError(fname, "x size of extend too large", HxGlobalError::HX_GE_INVALID);
56         return HxImageRep();
57     }
58     if ((begin.y() + img.sizes().y()) > background.sizes().y())
```

```
59     {
60         HxGlobalError::instance()->reportError(fname, "y size of extend too large", HxGlobalError::HX_GE_INVALID);
61         return HxImageRep();
62     }
63     if ((img.dimensionality() > 2) && ((begin.z() + img.sizes().z()) > background.sizes().z()))
64     {
65         HxGlobalError::instance()->reportError(fname, "z size of extend too large", HxGlobalError::HX_GE_INVALID);
66         return HxImageRep();
67     }
68
69     return img.extend(background, begin);
70 }
```

7.69 HxExtendVal.h File Reference

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

Functions

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

  Extension of domain.

7.69.1 Detailed Description

7.69.2 Function Documentation

7.69.2.1 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     HxString fname("HxExtendVal");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22
23     if (img.pixelDimensionality() == 2)
24     {
25         if ((background.tag() != HxValue::V2I) && (background.tag() != HxValue::V2D))
26         {
27             HxGlobalError::instance()->reportError(fname, "value dimensionality is not equal to pixel dimensionalitie
28             return HxImageRep();
29         }
30     }
31     if (img.pixelDimensionality() == 3)
32     {
33         if ((background.tag() != HxValue::V3I) && (background.tag() != HxValue::V3D))
```

```
34         {
35             HxGlobalError::instance()->reportError(fname, "value dimensionality is not equal to pixel dimensionalitie
36             return HxImageRep();
37         }
38     }
39
40     if (begin.x() < 0)
41     {
42         HxGlobalError::instance()->reportError(fname, "begin.x less then 0", HxGlobalError::HX_GE_INVALID);
43         return HxImageRep();
44     }
45     if (begin.y() < 0)
46     {
47         HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
48         return HxImageRep();
49     }
50     if (begin.z() < 0)
51     {
52         HxGlobalError::instance()->reportError(fname, "begin.z less then 0", HxGlobalError::HX_GE_INVALID);
53         return HxImageRep();
54     }
55     if ((begin.x() + img.sizes().x()) > newSize.x())
56     {
57         HxGlobalError::instance()->reportError(fname, "x size of extend too large", HxGlobalError::HX_GE_INVALID);
58         return HxImageRep();
59     }
60     if ((begin.y() + img.sizes().y()) > newSize.y())
61     {
62         HxGlobalError::instance()->reportError(fname, "y size of extend too large", HxGlobalError::HX_GE_INVALID);
63         return HxImageRep();
64     }
65     if ((begin.z() + img.sizes().z()) > newSize.z())
66     {
67         HxGlobalError::instance()->reportError(fname, "z size of extend too large", HxGlobalError::HX_GE_INVALID);
68         return HxImageRep();
69     }
70
71
72     return img.extend(newSize, background, begin);
73 }
```

7.70 HxFloor.h File Reference

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

Functions

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

  Floor.

7.70.1 Detailed Description

7.70.2 Function Documentation

7.70.2.1 HxImageRep L_HXIMAGEREP HxFloor (HxImageRep *im*)

Floor.

The function computes the floor (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoFloor** (p. 1224). The image functor instantiator : **HxInstantiatorFloor** (p. 881).

```
13 {
14     HxString fname("HxFloor");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("floor");
23 }
```

7.71 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.

7.71.1 Detailed Description

7.71.2 Function Documentation

7.71.2.1 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 }
```

7.71.2.2 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 }
```

7.71.2.3 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 }
```

7.71.2.4 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 }
```

7.71.2.5 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 }
```

7.71.2.6 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 }
```

7.72 HxFuncBpo.c File Reference

```
#include "HxFuncBpo.h"
#include "HxCategories.h"
```

Functions

- template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void **HxFuncBpo** (DstDataPtrT dstPtr, Src1DataPtrT src1Ptr, Src2DataPtrT src2Ptr, **HxSizes** dstSize, BpoT &bpo, **HxTagTransInVar** dummy)

  Translation invariant binary pixel operation.

- template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void **HxFuncBpo** (DstDataPtrT dstPtr, Src1DataPtrT src1Ptr, Src2DataPtrT src2Ptr, **HxSizes** dstSize, BpoT &bpo, **HxTagTransVar** dummy)

  Translation variant binary pixel operation.

- template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void **HxFuncBpoDispatch** (DstDataPtrT dstPtr, Src1DataPtrT src1Ptr, Src2DataPtrT src2Ptr, **HxSizes** dstSize, BpoT &bpo)

  Dispatch function for binary pixel operation.

7.72.1 Detailed Description

7.72.2 Function Documentation

7.72.2.1 template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void HxFuncBpo (DstDataPtrT *dstPtr*, Src1DataPtrT *src1Ptr*, Src2DataPtrT *src2Ptr*, HxSizes *dstSize*, BpoT & *bpo*, HxTagTransInVar *dummy*)

Translation invariant binary pixel operation.

```
26 {
27     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
28     while (--nPix >= 0)
29         dstPtr.writeIncX(bpo.doIt(src1Ptr.readIncX(), src2Ptr.readIncX()));
30 }
```

7.72.2.2 template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void HxFuncBpo (DstDataPtrT *dstPtr*, Src1DataPtrT *src1Ptr*, Src2DataPtrT *src2Ptr*, HxSizes *dstSize*, BpoT & *bpo*, HxTagTransVar *dummy*)

Translation variant binary pixel operation.

```
39 {
40     for (int z=0 ; z<dstSize.z() ; z++) {
41         for (int y=0 ; y<dstSize.y() ; y++) {
42             for (int x=0 ; x<dstSize.x() ; x++) {
43                 dstPtr.writeIncX(
```

```
44                      bpo.doIt(src1Ptr.readIncX(), src2Ptr.readIncX(), x, y, z));
45              }
46          }
47      }
48  }
```

7.72.2.3 template<class DstDataPtrT, class Src1DataPtrT, class Src2DataPtrT, class BpoT> void HxFuncBpoDispatch (DstDataPtrT *dstPtr*, Src1DataPtrT *src1Ptr*, Src2DataPtrT *src2Ptr*, HxSizes *dstSize*, BpoT & *bpo*)

Dispatch function for binary pixel operation.

Dispatch is based on the TransVarianceCategory category defined in BpoT. Calls **HxFuncBpo**(DstDataPtrT,Src1DataPtrT,Src2DataPtrT,HxSizes,BpoT&,HxTagTransInVar) (p. 152) or **HxFuncBpo**(DstDataPtrT,Src1DataPtrT,Src2DataPtrT,HxSizes,BpoT&,HxTagTransVar) (p. 152).

```
61  {
62      HxFuncBpo(
63          dstPtr, src1Ptr, src2Ptr, dstSize, bpo,
64          typename BpoT::TransVarianceCategory());
65  }
```

7.73 HxFuncExportExtra.c File Reference

```
#include "HxFuncExportExtra.h"
#include "HxCategories.h"
```

Row_variations

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtra_Row_OutTi** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, int nPix, ExportExtraT &exportOp)

  Row : translation invariant.

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtra_Row_OutTv** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, int nPix, ExportExtraT &exportOp, int x, int y, int z)

  Row : translation variant.

ExportExtra_variations

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtra** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, **HxSizes** sizes, ExportExtraT &exportOp, **HxTagTransInVar** dummy1, **HxTag1Phase** dummy2)

  Translation invariant, 1 phase export extra operation.

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtra** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, **HxSizes** sizes, ExportExtraT &exportOp, **HxTagTransInVar** dummy1, **HxTagNPhase** dummy2)

Translation invariant, n phase export extra operation.

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtra** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, **HxSizes** sizes, ExportExtraT &exportOp, **HxTagTransVar** dummy1, **HxTag1Phase** dummy2)

  Translation variant, 1 phase export extra operation.

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtra** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, **HxSizes** sizes, ExportExtraT &exportOp, **HxTagTransVar** dummy1, **HxTagNPhase** dummy2)

  Translation variant, n phase export extra operation.

Functions

- template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void **HxFuncExportExtraDispatch** (ImgDataPtrType imPtr, ExtraImgDataPtrType extraPtr, **HxSizes** sizes, ExportExtraT &exportOp)

  Dispatch function for HxFuncExportExtra (see **Global functions for ExportExtra** (p. **??**)*).*

7.73.1 Detailed Description

7.73.2 Function Documentation

7.73.2.1 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtra_Row_OutTi (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, int *nPix*, ExportExtraT & *exportOp*) [inline]

Row : translation invariant.

```
51 {
52     while (--nPix >= 0)
53         exportOp.doIt(imPtr.readIncX(), extraPtr.readIncX());
54 }
```

7.73.2.2 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtra_Row_OutTv (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, int *nPix*, ExportExtraT & *exportOp*, int *x*, int *y*, int *z*) [inline]

Row : translation variant.

```
61 {
62     while (--nPix >= 0)
63         exportOp.doIt(imPtr.readIncX(), extraPtr.readIncX(), x++, y, z);
64 }
```

7.73.2.3 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtra (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, HxSizes *sizes*, ExportExtraT & *exportOp*, HxTagTransInVar *dummy1*, HxTag1Phase *dummy2*)

Translation invariant, 1 phase export extra operation.

```
80 {
81     for (int z=0; z<sizes.z(); z++) {
82         for (int y=0; y<sizes.y(); y++) {
83             HxFuncExportExtra_Row_OutTi(imPtr, extraPtr, sizes.x(), exportOp);
84             imPtr.incY();
85             extraPtr.incY();
86         }
87         imPtr.decY(sizes.y());
88         imPtr.incZ();
89         extraPtr.decY(sizes.y());
90         extraPtr.incZ();
91     }
92 }
```

7.73.2.4 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtra (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, HxSizes *sizes*, ExportExtraT & *exportOp*, HxTagTransInVar *dummy1*, HxTagNPhase *dummy2*)

Translation invariant, n phase export extra operation.

```
101 {
102     for (int phase=1 ; phase<=exportOp.nrPhases() ; phase++) {
103         exportOp.init(phase);
104         for (int z=0; z<sizes.z(); z++) {
105                 for (int y=0; y<sizes.y(); y++) {
106                     HxFuncExportExtra_Row_OutTi(imPtr, extraPtr, sizes.x(),
107                                                 exportOp);
108                     imPtr.incY();
109                     extraPtr.incY();
110                 }
111                 imPtr.decY(sizes.y());
112                 imPtr.incZ();
113                 extraPtr.decY(sizes.y());
114                 extraPtr.incZ();
115         }
116         exportOp.done(phase);
117     }
118 }
```

7.73.2.5 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtra (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, HxSizes *sizes*, ExportExtraT & *exportOp*, HxTagTransVar *dummy1*, HxTag1Phase *dummy2*)

Translation variant, 1 phase export extra operation.

```
127 {
128     for (int z=0; z<sizes.z(); z++) {
129         for (int y=0; y<sizes.y(); y++) {
130             HxFuncExportExtra_Row_OutTv(imPtr, extraPtr, sizes.x(), exportOp,
131                                         0, y, z);
```

```
132             imPtr.incY();
133             extraPtr.incY();
134         }
135         imPtr.decY(sizes.y());
136         imPtr.incZ();
137         extraPtr.decY(sizes.y());
138         extraPtr.incZ();
139     }
140 }
```

7.73.2.6 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtra (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, HxSizes *sizes*, ExportExtraT & *exportOp*, HxTagTransVar *dummy1*, HxTagNPhase *dummy2*)

Translation variant, n phase export extra operation.

```
149 {
150     for (int phase=1 ; phase<=exportOp.nrPhases() ; phase++) {
151         exportOp.init(phase);
152         for (int z=0; z<sizes.z(); z++) {
153             for (int y=0; y<sizes.y(); y++) {
154                 HxFuncExportExtra_Row_OutTv(imPtr, extraPtr, sizes.x(),
155                                             exportOp, 0, y, z);
156                 imPtr.incY();
157                 extraPtr.incY();
158             }
159             imPtr.decY(sizes.y());
160             imPtr.incZ();
161             extraPtr.decY(sizes.y());
162             extraPtr.incZ();
163         }
164         exportOp.done(phase);
165     }
166 }
```

7.73.2.7 template<class ImgDataPtrType, class ExtraImgDataPtrType, class ExportExtraT> void HxFuncExportExtraDispatch (ImgDataPtrType *imPtr*, ExtraImgDataPtrType *extraPtr*, HxSizes *sizes*, ExportExtraT & *exportOp*)

Dispatch function for HxFuncExportExtra (see **Global functions for ExportExtra** (p. **??**)).

Dispatch is based on the categories defined in ExportExtraT.

```
179 {
180     HxFuncExportExtra(
181         imPtr, extraPtr, sizes, exportOp,
182         typename ExportExtraT::TransVarianceCategory(),
183         typename ExportExtraT::PhaseCategory());
184 }
```

7.74 HxFuncGenConv2d.c File Reference

```
#include "HxFuncGenConv2d.h"
```

Functions

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2d_rowpixfunc** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)
- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2d_norowpixfunc** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)
- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dDispatch** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp, bool rowpixfunc)

  Dispatch function for GenConv2d.

7.74.1 Detailed Description

7.74.2 Function Documentation

7.74.2.1 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dDispatch (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*, bool *rowpixfunc*)

Dispatch function for GenConv2d.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = kernelSize/2, srcPtr is at (IX0,IY0)

kernel Input image, IS = kernelSize, IBS = 0

```
79 {
80     if (rowpixfunc)
81         HxFuncGenConv2d_rowpixfunc(dstPtr, srcPtr, kernel,
82                                    dstSize, pixOp, redOp);
83     else
84         HxFuncGenConv2d_norowpixfunc(dstPtr, srcPtr, kernel,
85                                      dstSize, pixOp, redOp);
86 }
```

7.75 HxFuncGenConv2dK1d.c File Reference

```
#include "HxFuncGenConv2dK1d.h"
#include "HxEnvironment.h"
```

Pix_variations

- template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_Line_XdirInp** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ArithT *ngbPtr, ArithT *kernel, int nPix, int kernelSize, PixOpT &pixOp, RedOpT &redOp, ArithT neutralElement)

Pix : X direction, inplace.

- template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_Line_YdirInp** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ArithT *ngbBuf, ArithT *ngbPtr, ArithT *kernel, int width, int kernelSize, PixOpT &pixOp, RedOpT &redOp, const ArithT &neutralElement)

  Pix : Y direction, inplace.

Line_variations

- template<class DstPtrT, class SrcPtrT, class ArithT, class PixOpT, class RedOpT, class KernelT> void **HxFuncGenConv2dK1d_Line_XdirSim** (DstPtrT dstPtr, SrcPtrT srcPtr, ArithT neutralElement, int nPix, int kernelSize, PixOpT &pixOp, RedOpT &redOp, KernelT &kernel)

  Line : X direction, simple.

- template<class DstPtrT, class SrcPtrT, class ArithT, class PixOpT, class RedOpT, class KernelT> void **HxFuncGenConv2dK1d_Line_YdirSim** (DstPtrT dstPtr, SrcPtrT srcPtr, ArithT neutralElement, int nPix, int kernelSize, PixOpT &pixOp, RedOpT &redOp, KernelT &kernel)

  Line : Y direction, simple.

- template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_Im_XdirInp** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ArithT *kernel, int width, int height, int kernelSize, PixOpT &pixOp, RedOpT &redOp)

  Line : X direction, inplace.

- template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_Im_YdirInp** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ArithT *kernel, **HxSizes** dstSize, int kernelSize, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, inplace.

GenConv2dK1d_variations

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_XdirSim** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv2dK1d : X direction, simple.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_YdirSim** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv2dK1d : Y direction, simple.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_XdirInp** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv2dK1d : X direction, inplace.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1d_YdirInp** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv2dK1d : Y direction, inplace.

Functions

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dK1dDispatch** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp, int dimension, bool inplace)

  Dispatch function for GenConv2dK1d (see **Global functions for GenConv2dK1d** (p. **??**)*) Dispatch is based on dimension and inplace parameters.*

7.75.1 Detailed Description

7.75.2 Function Documentation

7.75.2.1 template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_Line_XdirInp (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ArithT * *ngbPtr*, ArithT * *kernel*, int *nPix*, int *kernelSize*, PixOpT & *pixOp*, RedOpT & *redOp*, ArithT *neutralElement*) [inline, static]

Pix : X direction, inplace.

```
54 {
55     ArithT* ngbEnd = &ngbPtr[nPix];
56     for (;ngbPtr < ngbEnd; ngbPtr++)
57     {
58         ArithT result(neutralElement);
59         for (int k=0; k<kernelSize; k++)
60             redOp.doIt(result, pixOp.doIt(ngbPtr[k], kernel[k]));
61         dstPtr.writeIncX(result);
62         ngbPtr[kernelSize] = srcPtr.readIncX();
63     }
64 }
```

7.75.2.2 template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_Line_YdirInp (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ArithT * *ngbBuf*, ArithT * *ngbPtr*, ArithT * *kernel*, int *width*, int *kernelSize*, PixOpT & *pixOp*, RedOpT & *redOp*, const ArithT & *neutralElement*) [static]

Pix : Y direction, inplace.

```
75 {
76     int     lastKerElt = kernelSize-1;
77     int     ngbBufSize = width * kernelSize;
78     ArithT* ngbEnd = ngbBuf + ngbBufSize;
79
80     while (--width >= 0)
81     {
82         ngbPtr[lastKerElt] = srcPtr.readIncX();
```

```
83         ArithT result(neutralElement);
84         for (int k=0; k<kernelSize; k++)
85             redOp.doIt(result, pixOp.doIt(ngbPtr[k], kernel[k]));
86         dstPtr.writeIncX(result);
87         ngbPtr += kernelSize;
88         if (ngbPtr >= ngbEnd)
89             ngbPtr -= ngbBufSize;
90     }
91 }
```

7.75.2.3 template<class DstPtrT, class SrcPtrT, class ArithT, class PixOpT, class RedOpT, class KernelT> void HxFuncGenConv2dK1d_Line_XdirSim (DstPtrT *dstPtr*, SrcPtrT *srcPtr*, ArithT *neutralElement*, int *nPix*, int *kernelSize*, PixOpT & *pixOp*, RedOpT & *redOp*, KernelT & *kernel*) [static]

Line : X direction, simple.

```
107 {
108     while (--nPix >= 0) {
109         SrcPtrT nPtr(srcPtr);
110         srcPtr.incX();
111         ArithT result(neutralElement);
112         for (int i=0 ; i<kernelSize ; i++)
113             redOp.doIt(result, pixOp.doIt(nPtr.readIncX(), kernel(i)));
114         dstPtr.writeIncX(result);
115     }
116 }
```

7.75.2.4 template<class DstPtrT, class SrcPtrT, class ArithT, class PixOpT, class RedOpT, class KernelT> void HxFuncGenConv2dK1d_Line_YdirSim (DstPtrT *dstPtr*, SrcPtrT *srcPtr*, ArithT *neutralElement*, int *nPix*, int *kernelSize*, PixOpT & *pixOp*, RedOpT & *redOp*, KernelT & *kernel*) [static]

Line : Y direction, simple.

```
125 {
126     while (--nPix >= 0) {
127         SrcPtrT nPtr(srcPtr);
128         srcPtr.incX();
129         ArithT result(neutralElement);
130         for (int i=0 ; i<kernelSize ; i++) {
131             redOp.doIt(result, pixOp.doIt(nPtr.read(), kernel(i)));
132             nPtr.incY();
133         }
134         dstPtr.writeIncX(result);
135     }
136 }
```

7.75.2.5 template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_Im_XdirInp (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ArithT * *kernel*, int *width*, int *height*, int *kernelSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X direction, inplace.

```
145 {
146     HxPixelAllocator<ArithT> allocator;
147
148     ArithT*     ngbBuf = allocator.allocate(width + kernelSize);
149
150     srcPtr.decX(kernelSize/2);
151     while (--height >= 0)
152     {
153         for (int k=0; k<kernelSize; k++)
154             ngbBuf[k] = srcPtr.readIncX();
155         HxFuncGenConv2dK1d_Line_XdirInp(
156             dstPtr, srcPtr, ngbBuf,
157             kernel, width, kernelSize,
158             pixOp, redOp, RedOpT::neutralElement());
159         dstPtr.incY();
160         srcPtr.incXYZ(-kernelSize, 1);
161     }
162
163     allocator.deallocate(ngbBuf, width + kernelSize);
164 }
```

7.75.2.6 template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_Im_YdirInp (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ArithT * *kernel*, HxSizes *dstSize*, int *kernelSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, inplace.

```
173 {
174     HxPixelAllocator<ArithT> allocator;
175
176     int         ngbBufSize = dstSize.x() * kernelSize;
177     ArithT*     ngbBuf = allocator.allocate(ngbBufSize+kernelSize);
178     ArithT*     ngbPtr;
179     ArithT*     ngbEnd = ngbBuf + ngbBufSize;
180     SrcDataPtrT sPtr(srcPtr);
181     int         x, y, k, i;
182     ArithT*     ngbOverflow;
183
184     // initialize neighborhood buffer
185
186     sPtr.decY(kernelSize/2);
187     for (y=0; y<kernelSize-1; y++)
188     {
189         ngbPtr = &ngbBuf[y];
190         for (x=0; x<dstSize.x(); x++)
191         {
192             *ngbPtr = sPtr.readIncX();
193             ngbPtr += kernelSize;
194         }
195         sPtr.decX(dstSize.x());
196         sPtr.incY();
197     }
198     ngbOverflow = &ngbBuf[dstSize.x() * kernelSize];
199
200     // execute convolution by kernelsize clusters
201
202     srcPtr.incY(kernelSize/2);
203     ngbPtr = ngbBuf;
204
205     for (y=dstSize.y(); y>0; )
```

```
206     {
207         for (k = kernelSize < y ? kernelSize : y;--k >= 0; y--)
208         {
209             HxFuncGenConv2dK1d_Line_YdirInp(
210                 dstPtr, srcPtr, ngbBuf, ngbPtr,
211                 kernel, dstSize.x(), kernelSize,
212                 pixOp, redOp, RedOpT::neutralElement());
213             dstPtr.incY();
214             srcPtr.incY();
215             ngbPtr++;
216         }
217         // neighborhood buffer is cyclic
218         if (ngbPtr >= ngbEnd)
219             ngbPtr -= ngbBufSize;
220         // copy overflow neighborhood to first neighborhood
221         for (i=0; i<kernelSize; i++)
222             ngbBuf[i] = ngbOverflow[i];
223     }
224
225     // clean up
226
227     allocator.deallocate(ngbBuf, ngbBufSize+kernelSize);
228 }
```

7.75.2.7 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_XdirSim (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv2dK1d : X direction, simple.

```
244 {
245     HxSizes kerSize = kernel.sizes();
246     int kerWidth = kerSize.x();
247     int imgWidth = dstSize.x();
248     int imgHeight = dstSize.y();
249
250     typedef typename KernelT::ArithType ArithType;
251     ArithType neutralElement(RedOpT::neutralElement());
252
253     for (int y=0 ; y<imgHeight ; y++) {
254         DstDataPtrType dPtr(dstPtr);
255         dPtr.incY(y);
256         SrcDataPtrType sPtr(srcPtr);
257         sPtr.incXYZ(-(kerWidth/2), y);
258         HxFuncGenConv2dK1d_Line_XdirSim(
259             dPtr, sPtr, neutralElement, imgWidth, kerWidth,
260             pixOp, redOp, kernel);
261     }
262 }
```

7.75.2.8 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_YdirSim (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv2dK1d : Y direction, simple.

```
271 {
272     HxSizes kerSize = kernel.sizes();
273     int kerWidth = kerSize.x();
274     int imgWidth = dstSize.x();
275     int imgHeight = dstSize.y();
276 
277     typedef typename KernelT::ArithType ArithType;
278     ArithType neutralElement(RedOpT::neutralElement());
279 
280     for (int y=0 ; y<imgHeight ; y++) {
281         DstDataPtrType dPtr(dstPtr);
282         dPtr.incY(y);
283         SrcDataPtrType sPtr(srcPtr);
284         sPtr.incY(y-(kerWidth/2));
285         HxFuncGenConv2dK1d_Line_YdirSim(
286             dPtr, sPtr, neutralElement, imgWidth, kerWidth,
287             pixOp, redOp, kernel);
288     }
289 }
```

7.75.2.9 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_XdirInp (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv2dK1d : X direction, inplace.

```
298 {
299     HxSizes kerSize = kernel.sizes();
300     typedef typename KernelT::ArithType ArithType;
301 
302     HxPixelAllocator<ArithType> allocator;
303     ArithType* kernelBuffer = allocator.allocate(kerSize.x());
304     for (int i=0; i<kerSize.x(); i++)
305         kernelBuffer[i] = kernel(i);
306 
307     HxFuncGenConv2dK1d_Im_XdirInp(
308         dstPtr, srcPtr, kernelBuffer,
309         dstSize.x(), dstSize.y(), kerSize.x(), pixOp, redOp);
310 
311     allocator.deallocate(kernelBuffer, kerSize.x());
312 }
```

7.75.2.10 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1d_YdirInp (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv2dK1d : Y direction, inplace.

```
321 {
322     HxSizes kerSize = kernel.sizes();
323     typedef typename KernelT::ArithType ArithType;
324 
325     HxPixelAllocator<ArithType> allocator;
326     ArithType* kernelBuffer = allocator.allocate(kerSize.x());
327     for (int i=0; i<kerSize.x(); i++)
```

```
328         kernelBuffer[i] = kernel(i);
329
330     HxFuncGenConv2dK1d_Im_YdirInp(
331         dstPtr, srcPtr, kernelBuffer, dstSize, kerSize.x(), pixOp, redOp);
332
333     allocator.deallocate(kernelBuffer, kerSize.x());
334 }
```

7.75.2.11 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dK1dDispatch (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *dimension*, bool *inplace*)

Dispatch function for GenConv2dK1d (see **Global functions for GenConv2dK1d** (p. **??**)) Dispatch is based on dimension and inplace parameters.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = (?), IBS >= kernelSize/2, srcPtr is at (CIX0,CIY0)

kernel Input image, IS = kernelSize, IBS = 0

```
354 {
355     switch (dimension) {
356     case 1  :
357         if (inplace)
358             HxFuncGenConv2dK1d_XdirInp(
359                 dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
360         else
361             HxFuncGenConv2dK1d_XdirSim(
362                 dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
363         break;
364     case 2  :
365         if (inplace)
366             HxFuncGenConv2dK1d_YdirInp(
367                 dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
368         else
369             HxFuncGenConv2dK1d_YdirSim(
370                 dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
371         break;
372     default :
373         HxEnvironment::instance()->errorStream()
374             << "HxFuncGenConv2dK1dDispatch: "
375             << "cannot execute convolution in dimension " << dimension
376             << STD_ENDL;
377         HxEnvironment::instance()->flush();
378     }
379 }
```

7.76 HxFuncGenConv2dSep.c File Reference

```
#include "HxFuncGenConv2dSep.h"
#include "HxEnvironment.h"
```

Pix_variations

- template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Pix_Xdir** (ArithT *buf, SrcDataPtrT srcPtr, ArithT *kernel, int bufIdx, int kerWidth, PixOpT &pixOp, RedOpT &redOp, ArithT neutralElement)

  Pix : X direction.

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Pix_Ydir** (DstDataPtrT dstPtr, ArithT *bufPtr, ArithT *kernel, int bufWidth, int kerWidth, PixOpT &pixOp, RedOpT &redOp, ArithT neutralElement)

  Pix : Y direction.

Line_variations

- template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_Xdir** (ArithT *bufLine, SrcDataPtrT srcPtr, ArithT *kernel, int srcWidth, int dstWidth, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : X direction.

- template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_XdirInc** (ArithT *bufLine, SrcDataPtrT srcPtr, ArithT *kernel, int srcWidth, int dstWidth, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : X direction (inc).

- template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_XdirVerInc** (ArithT *buf, SrcDataPtrT srcPtr, ArithT *kernel1, int srcWidth, int dstWidth, int ker1Width, int ker2Width, PixOpT &pixOp, RedOpT &redOp)

  Line : X direction, "vertical buffer" (inc).

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_YdirNaiInc** (DstDataPtrT dstPtr, ArithT *buf, ArithT *kernel, int dstWidth, int dstHeight, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, naive (inc).

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_YdirSim** (DstDataPtrT dstPtr, ArithT *buf, ArithT *kernel, int dstWidth, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, simple.

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_YdirSimInc** (DstDataPtrT dstPtr, ArithT *buf, ArithT *kernel, int dstWidth, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, simple (inc).

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_YdirHor** (DstDataPtrT dstPtr, ArithT *buf, ArithT *kernel, int dstWidth, int lineIdx, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, "horizontal buffer".

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_YdirHorInc** (DstDataPtrT dstPtr, ArithT *buf, ArithT *kernel, int dstWidth, int lineIdx, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, "horizontal buffer" (inc).

- template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_YdirVerInc** (DstDataPtrT dstPtr, ArithT *buf, ArithT *kernel, int dstWidth, int lineIdx, int kerWidth, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, "vertical buffer" (inc).

- template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_XYdirVerCycInc** (DstDataPtrT dstPtr, ArithT *buf, ArithT *bufPtr, SrcDataPtrT srcPtr, ArithT *kernel1, ArithT *kernel2, int dstWidth, int ker1Width, int ker2Width, PixOpT &pixOp, RedOpT &redOp)

  Line : X and Y direction, "vertical buffer", two-way cyclic (inc).

- template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Line_XYdirMinInc** (DstDataPtrT dstPtr, ArithT *buf, SrcDataPtrT srcPtr, ArithT *kernel1, ArithT *kernel2, int srcWidth, int dstWidth, int ker1Width, int ker2Width, PixOpT &pixOp, RedOpT &redOp)

  Line : X and Y direction, "minimal buffer" (inc).

GenConv2dSep_variations

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Sim** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel1, KernelT &kernel2, **HxSizes** dstSize, **HxSizes** srcSize, PixOpT &pixOp, RedOpT &redOp, int vType)

  GenConv2dSep : simple.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Hor** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel1, KernelT &kernel2, **HxSizes** dstSize, **HxSizes** srcSize, PixOpT &pixOp, RedOpT &redOp, int vType)

  GenConv2dSep : "horizontal buffer".

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Ver** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel1, KernelT &kernel2, **HxSizes** dstSize, **HxSizes** srcSize, PixOpT &pixOp, RedOpT &redOp, int vType)

  GenConv2dSep : "vertical buffer".

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_VerCyc** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel1, KernelT &kernel2, **HxSizes** dstSize, **HxSizes** srcSize, PixOpT &pixOp, RedOpT &redOp, int vType)

  GenConv2dSep : "vertical buffer", two-way cyclic (localized computation).

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSep_Min** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT

&kernel1, KernelT &kernel2, **HxSizes** dstSize, **HxSizes** srcSize, PixOpT &pixOp, RedOpT &redOp, int vType)

GenConv2dSep : "minimal buffer".

Functions

- template<class KernelT, class ArithType> ArithType * **HxFuncGenConv2dSep_CopyKernel** (KernelT &kernel, ArithType)

  Copy (1d) kernel to an array of ArithT elements.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv2dSepDispatch** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel1, KernelT &kernel2, **HxSizes** dstSize, **HxSizes** srcSize, PixOpT &pixOp, RedOpT &redOp, int vType)

  Dispatch function for GenConv2dSep (see **Global functions for GenConv2dSep** (p. **??**)*) Dispatch is based on the vType parameter.*

7.76.1 Detailed Description

7.76.2 Function Documentation

7.76.2.1 template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Pix_Xdir (ArithT * *buf*, SrcDataPtrT *srcPtr*, ArithT * *kernel*, int *bufIdx*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*, ArithT *neutralElement*) [inline, static]

Pix : X direction.

Does a single genconv between srcPtr and kernel of kerWidth pixels and stores the result at buf[bufIdx]. Memory layout of all srcPtr and kernel is assumed contiguous.

```
82 {
83     ArithT result(neutralElement);
84     for (int k=0; k<kerWidth; k++)
85         redOp.doIt(result, pixOp.doIt(srcPtr.readIncX(), kernel[k]));
86     buf[bufIdx] = result;
87 }
```

7.76.2.2 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Pix_Ydir (DstDataPtrT *dstPtr*, ArithT * *bufPtr*, ArithT * *kernel*, int *bufWidth*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*, ArithT *neutralElement*) [inline, static]

Pix : Y direction.

Does a single genconv between bufPtr and kernel of kerWidth pixels and stores the result at dstPtr. Memory layout of kernel is assumed to be contiguous. Pixels in buf are assumed to be bufWidth elements apart.

```
103 {
104     ArithT result(neutralElement);
```

```
105     int idx = 0;
106     for (int k=0; k<kerWidth; k++) {
107         redOp.doIt(result, pixOp.doIt(bufPtr[idx], kernel[k]));
108         idx += bufWidth;
109     }
110     dstPtr.write(result);
111 }
```

7.76.2.3 template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_Xdir (ArithT * *bufLine*, SrcDataPtrT *srcPtr*, ArithT * *kernel*, int *srcWidth*, int *dstWidth*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X direction.

Processes an entire line starting at srcPtr. Calls HxFuncGenConv2dSep_Pix_Xdir "dstWidth" times to do a neighbourhood. The resulting "dstWidth" values are stored on bufLine (contiguous).

```
132 {
133     for (int x=0 ; x<dstWidth ; x++) {
134         HxFuncGenConv2dSep_Pix_Xdir(
135             bufLine, srcPtr, kernel, x, kerWidth,
136             pixOp, redOp, RedOpT::neutralElement());
137         srcPtr.incX();
138     }
139 }
```

7.76.2.4 template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_XdirInc (ArithT * *bufLine*, SrcDataPtrT *srcPtr*, ArithT * *kernel*, int *srcWidth*, int *dstWidth*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X direction (inc).

Processes an entire line starting at srcPtr (including the "Pix" variation). The resulting "dstWidth" values are stored in bufLine (contiguous).

This function does the same operation as HxFuncGenConv2dSep_Line_Xdir but the code of HxFuncGenConv2dSep_Pix_Xdir is inserted in the loop (instead of calling the function like HxFuncGenConv2dSep_Line_Xdir does).

```
156 {
157     ArithT neutralElement = RedOpT::neutralElement();
158     for (int x=0 ; x<dstWidth ; x++) {
159         SrcDataPtrT sPtr(srcPtr);
160         ArithT result(neutralElement);
161         for (int k=0; k<kerWidth; k++)
162             redOp.doIt(result, pixOp.doIt(sPtr.readIncX(), kernel[k]));
163         bufLine[x] = result;
164         srcPtr.incX();
165     }
166 }
```

7.76.2.5 template<class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_XdirVerInc (ArithT * *buf*, SrcDataPtrT *srcPtr*, ArithT * *kernel1*, int *srcWidth*, int *dstWidth*, int *ker1Width*, int *ker2Width*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X direction, "vertical buffer" (inc).

Processes an entire line starting at srcPtr (including the "Pix" variation). The resulting "dstWidth" values are stored in buf using a stride ker2Width.

```
180 {
181     ArithT neutralElement = RedOpT::neutralElement();
182     int idx = 0;
183     for (int x=0 ; x<dstWidth ; x++) {
184         SrcDataPtrT sPtr(srcPtr);
185         ArithT result(neutralElement);
186         for (int k=0; k<ker1Width; k++)
187             redOp.doIt(result, pixOp.doIt(sPtr.readIncX(), kernel1[k]));
188         buf[idx] = result;
189         idx += ker2Width;
190         srcPtr.incX();
191     }
192 }
```

7.76.2.6 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_YdirNaiInc (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *kernel*, int *dstWidth*, int *dstHeight*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, naive (inc).

```
201 {
202     ArithT neutralElement = RedOpT::neutralElement();
203     for (int y=0 ; y<dstHeight ; y++) {
204         ArithT result(neutralElement);
205         int idx = 0;
206         for (int k=0; k<kerWidth; k++) {
207             redOp.doIt(result, pixOp.doIt(buf[idx], kernel[k]));
208             idx += dstWidth;
209         }
210         dstPtr.write(result);
211         dstPtr.incY();
212         buf += dstWidth;
213     }
214 }
```

7.76.2.7 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_YdirSim (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *kernel*, int *dstWidth*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, simple.

```
223 {
224     for (int x=0 ; x<dstWidth ; x++) {
225         ArithT* bPtr = &buf[x];
226         HxFuncGenConv2dSep_Pix_Ydir(
```

```
227             dstPtr, bPtr, kernel, dstWidth, kerWidth,
228             pixOp, redOp, RedOpT::neutralElement());
229         dstPtr.incX();
230     }
231 }
```

7.76.2.8 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_YdirSimInc (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *kernel*, int *dstWidth*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, simple (inc).

```
240 {
241     ArithT neutralElement = RedOpT::neutralElement();
242     for (int x=0 ; x<dstWidth ; x++) {
243         ArithT* bufPtr = &buf[x];
244         ArithT result(neutralElement);
245         int idx = 0;
246         for (int k=0; k<kerWidth; k++) {
247             redOp.doIt(result, pixOp.doIt(bufPtr[idx], kernel[k]));
248             idx += dstWidth;
249         }
250         dstPtr.writeIncX(result);
251     }
252 }
```

7.76.2.9 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_YdirHor (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *kernel*, int *dstWidth*, int *lineIdx*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, "horizontal buffer".

```
261 {
262     HxPixelAllocator<ArithT> allocator;
263 
264     // Copy kernel data into cycKer to match the cycle of buf
265 
266     ArithT* cycKer = allocator.allocate(kerWidth);
267     for (int k=0 ; k<kerWidth ; k++) {
268         cycKer[lineIdx] = kernel[k];
269         lineIdx = (lineIdx + 1) % kerWidth;
270     }
271 
272     for (int x=0 ; x<dstWidth ; x++) {
273         ArithT* bufPtr = &buf[x];
274         HxFuncGenConv2dSep_Pix_Ydir(
275             dstPtr, bufPtr, cycKer, dstWidth, kerWidth,
276             pixOp, redOp, RedOpT::neutralElement());
277         dstPtr.incX();
278     }
279 
280     allocator.deallocate(cycKer, kerWidth);
281 }
```

7.76.2.10 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_YdirHorInc (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *kernel*, int *dstWidth*, int *lineIdx*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, "horizontal buffer" (inc).

```
290 {
291     HxPixelAllocator<ArithT> allocator;
292 
293     // Copy kernel data into cycKer to match the cycle of buf
294 
295     ArithT* cycKer = allocator.allocate(kerWidth);
296     for (int k=0 ; k<kerWidth ; k++) {
297         cycKer[lineIdx] = kernel[k];
298         lineIdx = (lineIdx + 1) % kerWidth;
299     }
300 
301     ArithT neutralElement = RedOpT::neutralElement();
302     for (int x=0 ; x<dstWidth ; x++) {
303         ArithT* bufPtr = &buf[x];
304         ArithT result(neutralElement);
305         int idx = 0;
306         for (int k=0; k<kerWidth; k++) {
307             redOp.doIt(result, pixOp.doIt(bufPtr[idx], cycKer[k]));
308             idx += dstWidth;
309         }
310         dstPtr.writeIncX(result);
311     }
312 
313     allocator.deallocate(cycKer, kerWidth);
314 }
```

7.76.2.11 template<class DstDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_YdirVerInc (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *kernel*, int *dstWidth*, int *lineIdx*, int *kerWidth*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, "vertical buffer" (inc).

```
323 {
324     HxPixelAllocator<ArithT> allocator;
325 
326     // Copy kernel data into cycKer to match the cycle of buf
327 
328     ArithT* cycKer = allocator.allocate(kerWidth);
329     for (int k=0 ; k<kerWidth ; k++) {
330         cycKer[lineIdx] = kernel[k];
331         lineIdx = (lineIdx + 1) % kerWidth;
332     }
333 
334     ArithT neutralElement = RedOpT::neutralElement();
335     int idx = 0;
336     for (int x=0 ; x<dstWidth ; x++) {
337         ArithT* bufPtr = &buf[idx];
338         ArithT result(neutralElement);
339         for (int k=0; k<kerWidth; k++) {
340             redOp.doIt(result, pixOp.doIt(bufPtr[k], cycKer[k]));
341         }
342         dstPtr.writeIncX(result);
```

```
343         idx += kerWidth;
344     }
345
346     allocator.deallocate(cycKer, kerWidth);
347 }
```

7.76.2.12 template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_XYdirVerCycInc (DstDataPtrT *dstPtr*, ArithT * *buf*, ArithT * *bufPtr*, SrcDataPtrT *srcPtr*, ArithT * *kernel1*, ArithT * *kernel2*, int *dstWidth*, int *ker1Width*, int *ker2Width*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X and Y direction, "vertical buffer", two-way cyclic (inc).

```
358 {
359     ArithT  neutralElement = RedOpT::neutralElement();
360     int     lastKerElt = ker2Width-1;
361     int     bufCycSize = dstWidth * ker2Width;
362     ArithT* bufOverflow = buf + bufCycSize;
363
364     while (--dstWidth >= 0) {
365         // first do the X direction (from src to buf) for the last element
366         // needed for the kernel in the Y direction (all other elements
367         // needed for the Y direction are already computed and stored in buf)
368         SrcDataPtrT sPtr(srcPtr);
369         ArithT result1(neutralElement);
370         for (int k1=0 ; k1<ker1Width ; k1++)
371             redOp.doIt(result1, pixOp.doIt(sPtr.readIncX(), kernel1[k1]));
372         bufPtr[lastKerElt] = result1;
373         srcPtr.incX();
374         // now the Y direction from buf to dst
375         ArithT result2(neutralElement);
376         for (int k2=0 ; k2<ker2Width ; k2++)
377             redOp.doIt(result2, pixOp.doIt(bufPtr[k2], kernel2[k2]));
378         dstPtr.writeIncX(result2);
379         bufPtr += ker2Width;
380         if (bufPtr >= bufOverflow)
381             bufPtr -= bufCycSize;
382     }
383 }
```

7.76.2.13 template<class DstDataPtrT, class SrcDataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Line_XYdirMinInc (DstDataPtrT *dstPtr*, ArithT * *buf*, SrcDataPtrT *srcPtr*, ArithT * *kernel1*, ArithT * *kernel2*, int *srcWidth*, int *dstWidth*, int *ker1Width*, int *ker2Width*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X and Y direction, "minimal buffer" (inc).

```
431 {
432     ArithT neutralElement = RedOpT::neutralElement();
433
434     // first Y dir from src to buf using kernel2
435
436     for (int x=0 ; x<srcWidth ; x++) {
437         SrcDataPtrT sPtr(srcPtr);
438         ArithT result(neutralElement);
439         for (int k=0 ; k<ker2Width ; k++) {
```

```
440                 redOp.doIt(result, pixOp.doIt(sPtr.read(), kernel2[k]));
441                 sPtr.incY();
442         }
443         buf[x] = result;
444         srcPtr.incX();
445     }
446
447     // now X dir from buf to dst using kernel1
448
449     for (int x2=0 ; x2<dstWidth ; x2++) {
450         ArithT* bufPtr = &buf[x2];
451         ArithT result(neutralElement);
452         for (int k=0; k<ker1Width; k++)
453             redOp.doIt(result, pixOp.doIt(bufPtr[k], kernel1[k]));
454         dstPtr.writeIncX(result);
455     }
456 }
```

7.76.2.14 template<class KernelT, class ArithType> ArithType* HxFuncGenConv2dSep_CopyKernel (KernelT & *kernel*, ArithType) [static]

Copy (1d) kernel to an array of ArithT elements.

```
466 {
467     HxPixelAllocator<ArithType> allocator;
468     HxSizes kerSize = kernel.sizes();
469     int width = kerSize.x();
470     ArithType* kerArray = allocator.allocate(width);
471     for (int i=0; i<width; i++)
472         kerArray[i] = kernel(i);
473     return kerArray;
474 }
```

7.76.2.15 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Sim (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel1*, KernelT & *kernel2*, HxSizes *dstSize*, HxSizes *srcSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *vType*) [static]

GenConv2dSep : simple.

The conceptually simple implementation basically does the X direction from src to a scratch image, and then the Y direction from the scratch image to dst.

src has a complete border (so both in the X and Y direction). scratch has a border in the Y direction only. dst has no border.

Has two basic versions (designated by vType). The versions do (conceptuelly) the same algorithm in the X direction. With vType == 0 the Y direction is processed in column order (so first the whole column with X=0 is processed before we move on to the next column. With vType == {1,2} the Y direction is processed in a row-wise manner. That is, the kernel is applied at the first position in all columns before it is applied to the next position (again in all columns).

Further variations determine whether the processing of a single kernel is done in a separate function or not.

```
510 {
511     typedef typename KernelT::ArithType ArithType;
512     HxPixelAllocator<ArithType> allocator;
```

```
513
514     int ker1Width = kernel1.sizes().x();
515     ArithType* ker1Array = HxFuncGenConv2dSep_CopyKernel(kernel1, ArithType());
516
517     int ker2Width = kernel2.sizes().x();
518     ArithType* ker2Array = HxFuncGenConv2dSep_CopyKernel(kernel2, ArithType());
519
520     int srcWidth = srcSize.x();
521     int srcHeight = srcSize.y();
522     int dstWidth = dstSize.x();
523     int dstHeight = dstSize.y();
524
525     // Allocate scratchImage of size (dstWidth,srcHeight)
526
527     int scratchSize = dstWidth * srcHeight;
528     ArithType* scratch = allocator.allocate(scratchSize);
529     ArithType* sLine;
530     int x, y;
531
532     // do the x direction from src to scratch
533
534     for (y=0 ; y<srcHeight ; y++) {
535         SrcDataPtrType sPtr(srcPtr);
536         sPtr.incY(y);
537         sLine = &scratch[y * dstWidth];
538         if (vType == 1)
539             HxFuncGenConv2dSep_Line_Xdir(
540                 sLine, sPtr, ker1Array, srcWidth, dstWidth, ker1Width,
541                 pixOp, redOp);
542         if ((vType == 0) || (vType == 2))
543             HxFuncGenConv2dSep_Line_XdirInc(
544                 sLine, sPtr, ker1Array, srcWidth, dstWidth, ker1Width,
545                 pixOp, redOp);
546     }
547
548     // do the y direction from scratch to dst
549
550     if (vType == 0) {
551         for (x=0 ; x<dstWidth ; x++) {
552             DstDataPtrType dPtr(dstPtr);
553             dPtr.incX(x);
554             sLine = &scratch[x];
555             HxFuncGenConv2dSep_Line_YdirNaiInc(
556                 dPtr, sLine, ker2Array, dstWidth, dstHeight, ker2Width,
557                 pixOp, redOp);
558         }
559     } else { // vType == 1 or 2
560         for (y=0 ; y<dstHeight ; y++) {
561             DstDataPtrType dPtr(dstPtr);
562             dPtr.incY(y);
563             sLine = &scratch[y * dstWidth];
564             if (vType == 1)
565                 HxFuncGenConv2dSep_Line_YdirSim(
566                     dPtr, sLine, ker2Array, dstWidth, ker2Width, pixOp, redOp);
567             if (vType == 2)
568                 HxFuncGenConv2dSep_Line_YdirSimInc(
569                     dPtr, sLine, ker2Array, dstWidth, ker2Width, pixOp, redOp);
570         }
571     }
572
573     allocator.deallocate(ker1Array, ker1Width);
574     allocator.deallocate(ker2Array, ker2Width);
575     allocator.deallocate(scratch, scratchSize);
576 }
```

7.76.2.16 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Hor (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel1*, KernelT & *kernel2*, HxSizes *dstSize*, HxSizes *srcSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *vType*) [static]

GenConv2dSep : "horizontal buffer".

This implementation does the X direction from src to a "horizontal buffer", and then the Y direction from the buffer to dst.

Processing the image is done in a row-wise manner. The buffer holds just enough (processed) rows of the image to be able to apply the kernel in the Y direction. So, the buffer is used in a cyclic manner.

The buffer is "horizontal" in that the data organization matches that of the images. That is, if pixels are next to eachother on a row in src, their processed results are also next to eachother in the buffer. The size of the buffer is dstWidth x ker2Width.

src has a complete border (so both in the X and Y direction). the buffer has no border. dst has no border.

Has two (designated by vType) that determine whether the processing of a single kernel is done in a separate function or not.

```
607 {
608     typedef typename KernelT::ArithType ArithType;
609     HxPixelAllocator<ArithType> allocator;
610 
611     int ker1Width = kernel1.sizes().x();
612     ArithType* ker1Array = HxFuncGenConv2dSep_CopyKernel(kernel1, ArithType());
613 
614     int ker2Width = kernel2.sizes().x();
615     ArithType* ker2Array = HxFuncGenConv2dSep_CopyKernel(kernel2, ArithType());
616 
617     int srcWidth = srcSize.x();
618     int srcHeight = srcSize.y();
619     int dstWidth = dstSize.x();
620     int dstHeight = dstSize.y();
621 
622     // Allocate buffer of size (dstWidth,ker2Width)
623 
624     int bufSize = dstWidth * ker2Width;
625     ArithType* buf = allocator.allocate(bufSize);
626     ArithType* bufLine;
627     int lineIdx = 0; // (cyclic) line index in buf
628 
629     // initialize buf with first ker2Width - 1 lines
630     int y;
631     for (y=0 ; y<ker2Width-1 ; y++) {
632         bufLine = &buf[lineIdx * dstWidth];
633         if (vType == 3)
634             HxFuncGenConv2dSep_Line_Xdir(
635                 bufLine, srcPtr, ker1Array, srcWidth, dstWidth, ker1Width,
636                 pixOp, redOp);
637         if (vType == 4)
638             HxFuncGenConv2dSep_Line_XdirInc(
639                 bufLine, srcPtr, ker1Array, srcWidth, dstWidth, ker1Width,
640                 pixOp, redOp);
641         lineIdx = (lineIdx + 1) % ker2Width; // buffer is cyclic
642         srcPtr.incY();
643     }
644 
645     // now do the image
646 
647     for (y=0 ; y<dstHeight ; y++) {
648 
```

```
649         // do X direction to next line in the buffer
650         // the next line is actually the last "element" for ker2 in the Y direction
651
652         bufLine = &buf[lineIdx * dstWidth];
653         if (vType == 3)
654             HxFuncGenConv2dSep_Line_Xdir(
655                 bufLine, srcPtr, ker1Array, srcWidth, dstWidth, ker1Width,
656                 pixOp, redOp);
657         if (vType == 4)
658             HxFuncGenConv2dSep_Line_XdirInc(
659                 bufLine, srcPtr, ker1Array, srcWidth, dstWidth, ker1Width,
660                 pixOp, redOp);
661         lineIdx = (lineIdx + 1) % ker2Width; // buffer is cyclic
662         srcPtr.incY();
663
664         // do Y direction from buffer to dstImg
665         // Since the buffer is cyclic, the "ker2" location starts at lineIdx
666
667         if (vType == 3)
668             HxFuncGenConv2dSep_Line_YdirHor(
669                 dstPtr, buf, ker2Array, dstWidth, lineIdx, ker2Width,
670                 pixOp, redOp);
671         if (vType == 4)
672             HxFuncGenConv2dSep_Line_YdirHorInc(
673                 dstPtr, buf, ker2Array, dstWidth, lineIdx, ker2Width,
674                 pixOp, redOp);
675         dstPtr.incY();
676     }
677
678     allocator.deallocate(ker1Array, ker1Width);
679     allocator.deallocate(ker2Array, ker2Width);
680     allocator.deallocate(buf, bufSize);
681 }
```

7.76.2.17 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Ver (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel1*, KernelT & *kernel2*, HxSizes *dstSize*, HxSizes *srcSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *vType*) [static]

GenConv2dSep : ”vertical buffer”.

This implementation does the X direction from src to a ”vertical buffer”, and then the Y direction from the buffer to dst.

Processing the image is done in a row-wise manner. The buffer holds just enough (processed) rows of the image to be able to apply the kernel in the Y direction. So, the buffer is used in a cyclic manner.

The buffer is ”vertical” in that the data organization is rotated w.r.t. the images. That is, if pixels are next to eachother on a row in src, their processed results are ker2width apart in the buffer. In other words, the processing of a (horizontal) row in the image is stored in a (vertical) column of the buffer. The size of the buffer is ker2Width x dstWidth.

src has a complete border (so both in the X and Y direction). the buffer has no border. dst has no border.

Has two (designated by vType) that determine whether the processing of a single kernel is done in a separate function or not.

```
714 {
715     typedef typename KernelT::ArithType ArithType;
716     HxPixelAllocator<ArithType> allocator;
717
```

```
718     int ker1Width = kernel1.sizes().x();
719     ArithType* ker1Array = HxFuncGenConv2dSep_CopyKernel(kernel1, ArithType());
720
721     int ker2Width = kernel2.sizes().x();
722     ArithType* ker2Array = HxFuncGenConv2dSep_CopyKernel(kernel2, ArithType());
723
724     int srcWidth = srcSize.x();
725     int srcHeight = srcSize.y();
726     int dstWidth = dstSize.x();
727     int dstHeight = dstSize.y();
728
729     // Allocate buffer of size (ker2Width,dstWidth)
730
731     int bufSize = dstWidth * ker2Width;
732     ArithType* buf = allocator.allocate(bufSize);
733     ArithType* bufLine;
734     int lineIdx = 0; // (cyclic) line index in buf
735
736     // initialize buf with first ker2Width - 1 lines
737     int y;
738     for (y=0 ; y<ker2Width-1 ; y++) {
739         bufLine = &buf[lineIdx];
740         HxFuncGenConv2dSep_Line_XdirVerInc(
741             bufLine, srcPtr, ker1Array, srcWidth, dstWidth,
742             ker1Width, ker2Width, pixOp, redOp);
743         lineIdx = (lineIdx + 1) % ker2Width; // buffer is cyclic
744         srcPtr.incY();
745     }
746
747     // now do the image
748
749     for (y=0 ; y<dstHeight ; y++) {
750
751         // do X direction to next line in the buffer
752         // the next line is actually the last "element" for ker2 in the Y direction
753
754         bufLine = &buf[lineIdx];
755         HxFuncGenConv2dSep_Line_XdirVerInc(
756             bufLine, srcPtr, ker1Array, srcWidth, dstWidth,
757             ker1Width, ker2Width, pixOp, redOp);
758         lineIdx = (lineIdx + 1) % ker2Width; // buffer is cyclic
759         srcPtr.incY();
760
761         // do Y direction from buffer to dstImg
762         // Since the buffer is cyclic, the "ker2" location starts at lineIdx
763
764         HxFuncGenConv2dSep_Line_YdirVerInc(
765             dstPtr, buf, ker2Array, dstWidth, lineIdx, ker2Width,
766             pixOp, redOp);
767         dstPtr.incY();
768     }
769
770     allocator.deallocate(ker1Array, ker1Width);
771     allocator.deallocate(ker2Array, ker2Width);
772     allocator.deallocate(buf, bufSize);
773 }
```

7.76.2.18 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_VerCyc (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel1*, KernelT & *kernel2*, HxSizes *dstSize*, HxSizes *srcSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *vType*) [static]

GenConv2dSep : "vertical buffer", two-way cyclic (localized computation).

This implementation does the X direction from src to a "vertical buffer", and then the Y direction from the buffer to dst.

Processing the image is done in a row-wise manner. The buffer holds just enough (processed) rows of the image to be able to apply the kernel in the Y direction. So, the buffer is used in a cyclic manner.

The buffer is "vertical" in that the data organization is rotated w.r.t. the images. That is, if pixels are next to eachother on a row in src, their processed results are ker2width apart in the buffer. In other words, the processing of a (horizontal) row in the image is stored in a (vertical) column of the buffer. The size of the buffer is ker2Width x (dstWidth + 1). The 1 is for overflow.

In this variation, the buffer in not only cyclic in the "X direction" (that is when storing "row results" in columns) but it is also cyclic in the "Y direction". That is, a "row result" is not always stored at the beginning of a column of the buffer.

src has a complete border (so both in the X and Y direction). the buffer has no border. dst has no border.

```
808 {
809     typedef typename KernelT::ArithType ArithType;
810     HxPixelAllocator<ArithType> allocator;
811 
812     int ker1Width = kernel1.sizes().x();
813     ArithType* ker1Array = HxFuncGenConv2dSep_CopyKernel(kernel1, ArithType());
814 
815     int ker2Width = kernel2.sizes().x();
816     ArithType* ker2Array = HxFuncGenConv2dSep_CopyKernel(kernel2, ArithType());
817 
818     int srcWidth = srcSize.x();
819     int srcHeight = srcSize.y();
820     int dstWidth = dstSize.x();
821     int dstHeight = dstSize.y();
822 
823     // Allocate buffer of size (ker2Width, dstWidth+1)
824 
825     int         bufCycSize = ker2Width * dstWidth; // size of the "cyclic" part
826     int         bufTotSize = bufCycSize + ker2Width; // the total size
827     ArithType*  buf = allocator.allocate(bufTotSize);
828     ArithType*  bufOverflow = buf + bufCycSize;
829 
830     // initialize buf with first ker2Width - 1 lines
831 
832     int lineIdx = 0;
833     int y;
834     for (y=0 ; y<ker2Width-1 ; y++) {
835         ArithType *bufLine = &buf[lineIdx];
836         HxFuncGenConv2dSep_Line_XdirVerInc(
837             bufLine, srcPtr, ker1Array, srcWidth, dstWidth,
838             ker1Width, ker2Width, pixOp, redOp);
839         lineIdx++;
840         srcPtr.incY();
841     }
842 
843     // now do the image
844 
845     ArithType* bufPtr= buf;
846     for (y=dstHeight ; y>0 ; ) {
```

```
847         int k;
848         for (k=ker2Width < y ? ker2Width : y ; --k >= 0 ; y--) {
849             HxFuncGenConv2dSep_Line_XYdirVerCycInc(
850                 dstPtr, buf, bufPtr, srcPtr, ker1Array, ker2Array,
851                 dstWidth, ker1Width, ker2Width, pixOp, redOp);
852             dstPtr.incY();
853             srcPtr.incY();
854             bufPtr++;
855         }
856         // buffer is cyclic
857         if (bufPtr >= bufOverflow)
858             bufPtr -= bufCycSize;
859         // copy overflow of buffer to first line of buffer
860         for (k=0; k<ker2Width; k++)
861             buf[k] = bufOverflow[k];
862     }
863
864     allocator.deallocate(ker1Array, ker1Width);
865     allocator.deallocate(ker2Array, ker2Width);
866     allocator.deallocate(buf, bufTotSize);
867 }
```

7.76.2.19 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dSep_Min (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel1*, KernelT & *kernel2*, HxSizes *dstSize*, HxSizes *srcSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *vType*) [static]

GenConv2dSep : ”minimal buffer”.

This implementation does the Y (!) direction from src to a buffer, and then the X direction from the buffer to dst.

Processing the image is done in a row-wise manner. Since the Y direction is done first, the buffer needs to contain only one row of results. The size of the buffer is srcWidth.

src has a complete border (so both in the X and Y direction). the buffer has a border in the X direction only. dst has no border.

```
890 {
891     typedef typename KernelT::ArithType ArithType;
892     HxPixelAllocator<ArithType> allocator;
893
894     int ker1Width = kernel1.sizes().x();
895     ArithType* ker1Array = HxFuncGenConv2dSep_CopyKernel(kernel1, ArithType());
896
897     int ker2Width = kernel2.sizes().x();
898     ArithType* ker2Array = HxFuncGenConv2dSep_CopyKernel(kernel2, ArithType());
899
900     int srcWidth = srcSize.x();
901     int srcHeight = srcSize.y();
902     int dstWidth = dstSize.x();
903     int dstHeight = dstSize.y();
904
905     // Allocate buffer of size (srcWidth)
906
907     int bufSize = srcWidth;
908     ArithType* buf = allocator.allocate(bufSize);
909
910     // now do the image
911
912     for (int y=0 ; y<dstHeight ; y++) {
```

```
913         HxFuncGenConv2dSep_Line_XYdirMinInc(
914             dstPtr, buf, srcPtr, ker1Array, ker2Array, srcWidth, dstWidth,
915             ker1Width, ker2Width, pixOp, redOp);
916         srcPtr.incY();
917         dstPtr.incY();
918     }
919
920     allocator.deallocate(ker1Array, ker1Width);
921     allocator.deallocate(ker2Array, ker2Width);
922     allocator.deallocate(buf, bufSize);
923 }
```

7.76.2.20 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv2dSepDispatch (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel1*, KernelT & *kernel2*, HxSizes *dstSize*, HxSizes *srcSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *vType*)

Dispatch function for GenConv2dSep (see **Global functions for GenConv2dSep** (p. **??**)) Dispatch is based on the vType parameter.

Assertions:

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = (ker1_NBW,ker2_NBW), srcPtr is at (IX0,IY0)

ker1Ptr Input image, IS = ker1Size, IBS = 0

ker2Ptr Input image, IS = ker2Size, IBS = 0

```
948 {
949     switch (vType) {
950     case 0:
951     case 1:
952     case 2:
953         HxFuncGenConv2dSep_Sim(
954             dstPtr, srcPtr, kernel1, kernel2, dstSize, srcSize, pixOp, redOp, vType);
955         break;
956     case 3:
957     case 4:
958         HxFuncGenConv2dSep_Hor(
959             dstPtr, srcPtr, kernel1, kernel2, dstSize, srcSize, pixOp, redOp, vType);
960         break;
961     case 5:
962         HxFuncGenConv2dSep_Ver(
963             dstPtr, srcPtr, kernel1, kernel2, dstSize, srcSize, pixOp, redOp, vType);
964         break;
965     case 6: // this is the default (set in HxImgFtorGenConv2dSep)
966         HxFuncGenConv2dSep_VerCyc(
967             dstPtr, srcPtr, kernel1, kernel2, dstSize, srcSize, pixOp, redOp, vType);
968         break;
969     case 7:
970         HxFuncGenConv2dSep_Min(
971             dstPtr, srcPtr, kernel1, kernel2, dstSize, srcSize, pixOp, redOp, vType);
972         break;
973     default :
974         HxEnvironment::instance()->errorStream()
975             << "HxFuncGenConv2dSepDispatch: unknown vType " << vType
976             << STD_ENDL;
977         HxEnvironment::instance()->flush();
978     }
979 }
```

7.77 HxFuncGenConv3d.c File Reference

```
#include "HxFuncGenConv3d.h"
```

Functions

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3d_rowpixfunc** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)
- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3d_norowpixfunc** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)
- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3dDispatch** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp, bool rowpixfunc)

  Dispatch function for GenConv3d.

7.77.1 Detailed Description

7.77.2 Function Documentation

7.77.2.1 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv3dDispatch (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*, bool *rowpixfunc*)

Dispatch function for GenConv3d.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = kernelSize/2, srcPtr is at (IX0,IY0)

kernel Input image, IS = kernelSize, IBS = 0

```
88 {
89     if (rowpixfunc)
90         HxFuncGenConv3d_rowpixfunc(dstPtr, srcPtr, kernel,
91                                    dstSize, pixOp, redOp);
92     else
93         HxFuncGenConv3d_norowpixfunc(dstPtr, srcPtr, kernel,
94                                      dstSize, pixOp, redOp);
95 }
```

7.78 HxFuncGenConv3dK1d.c File Reference

```
#include "HxFuncGenConv3dK1d.h"
#include "HxEnvironment.h"
```

GenConv3dK1d_variations

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3dK1d_XdirSim** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv3dK1d : X direction, simple.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3dK1d_YdirSim** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv3dK1d : Y direction, simple.

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3dK1d_ZdirSim** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp)

  GenConv3dK1d : Z direction, simple.

Functions

- template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void **HxFuncGenConv3dK1dDispatch** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, KernelT &kernel, **HxSizes** dstSize, PixOpT &pixOp, RedOpT &redOp, int dimension)

  Dispatch function for GenConv3dK1d (see **Global functions for GenConv3dK1d** (p. **??**)*) Dispatch is based on dimension and inplace parameters.*

7.78.1 Detailed Description

7.78.2 Function Documentation

7.78.2.1 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv3dK1d_XdirSim (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv3dK1d : X direction, simple.

```
49 {
50     HxSizes kerSize = kernel.sizes();
51     int imgWidth = dstSize.x();
52     int imgHeight = dstSize.y();
53     int imgDepth = dstSize.z();
54     int kerWidth = kerSize.x();
55
56     int x, y, z, i;
57
58     typedef typename KernelT::ArithType ArithType;
59     ArithType result, tmpVal;
60
61     for (z=0 ; z<imgDepth ; z++) {
62         for (y=0 ; y<imgHeight ; y++) {
63             DstDataPtrType dPtr = dstPtr;
64             dPtr.incXYZ(0, y, z);
```

```
65             for (x=0 ; x<imgWidth ; x++) {
66                 SrcDataPtrType sPtr = srcPtr;
67                 sPtr.incXYZ(x, y, z);
68                 result = RedOpT::neutralElement();
69                 for (i=0 ; i<kerWidth ; i++) {
70                     tmpVal = pixOp.doIt(sPtr.read(), kernel(i));
71                     redOp.doIt(result, tmpVal);
72                     sPtr.incX();
73                 }
74                 dPtr.write(result);
75                 dPtr.incX();
76             }
77         }
78     }
79 }
```

7.78.2.2 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv3dK1d_YdirSim (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv3dK1d : Y direction, simple.

```
88 {
89     HxSizes kerSize = kernel.sizes();
90     int imgWidth = dstSize.x();
91     int imgHeight = dstSize.y();
92     int imgDepth = dstSize.z();
93     int kerWidth = kerSize.x();
94
95     int x, y, z, i;
96
97     typedef typename KernelT::ArithType ArithType;
98     ArithType result, tmpVal;
99
100     for (z=0 ; z<imgDepth ; z++) {
101         for (y=0 ; y<imgHeight ; y++) {
102             DstDataPtrType dPtr = dstPtr;
103             dPtr.incXYZ(0, y, z);
104             for (x=0 ; x<imgWidth ; x++) {
105                 SrcDataPtrType sPtr = srcPtr;
106                 sPtr.incXYZ(x, y, z);
107                 result = RedOpT::neutralElement();
108                 for (i=0 ; i<kerWidth ; i++) {
109                     tmpVal = pixOp.doIt(sPtr.read(), kernel(i));
110                     redOp.doIt(result, tmpVal);
111                     sPtr.incY();
112                 }
113                 dPtr.write(result);
114                 dPtr.incX();
115             }
116         }
117     }
118 }
```

7.78.2.3 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv3dK1d_ZdirSim (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

GenConv3dK1d : Z direction, simple.

```
127 {
128     HxSizes kerSize = kernel.sizes();
129     int imgWidth = dstSize.x();
130     int imgHeight = dstSize.y();
131     int imgDepth = dstSize.z();
132     int kerWidth = kerSize.x();
133
134     int x, y, z, i;
135
136     typedef typename KernelT::ArithType ArithType;
137     ArithType result, tmpVal;
138
139     for (z=0 ; z<imgDepth ; z++) {
140         for (y=0 ; y<imgHeight ; y++) {
141             DstDataPtrType dPtr = dstPtr;
142             dPtr.incXYZ(0, y, z);
143             for (x=0 ; x<imgWidth ; x++) {
144                 SrcDataPtrType sPtr = srcPtr;
145                 sPtr.incXYZ(x, y, z);
146                 result = RedOpT::neutralElement();
147                 for (i=0 ; i<kerWidth ; i++) {
148                     tmpVal = pixOp.doIt(sPtr.read(), kernel(i));
149                     redOp.doIt(result, tmpVal);
150                     sPtr.incZ();
151                 }
152                 dPtr.write(result);
153                 dPtr.incX();
154             }
155         }
156     }
157 }
```

7.78.2.4 template<class DstDataPtrType, class SrcDataPtrType, class KernelT, class PixOpT, class RedOpT> void HxFuncGenConv3dK1dDispatch (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, KernelT & *kernel*, HxSizes *dstSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *dimension*)

Dispatch function for GenConv3dK1d (see **Global functions for GenConv3dK1d** (p. **??**)) Dispatch is based on dimension and inplace parameters.

```
172 {
173     switch (dimension) {
174     case 1  :
175         HxFuncGenConv3dK1d_XdirSim(
176             dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
177         break;
178     case 2  :
179         HxFuncGenConv3dK1d_YdirSim(
180             dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
181         break;
182     case 3  :
183         HxFuncGenConv3dK1d_ZdirSim(
```

```
184             dstPtr, srcPtr, kernel, dstSize, pixOp, redOp);
185         break;
186     default :
187         HxEnvironment::instance()->errorStream()
188             << "HxFuncGenConv3dK1dDispatch: "
189             << "cannot execute convolution in dimension " << dimension
190             << STD_ENDL;
191         HxEnvironment::instance()->flush();
192     }
193 }
```

7.79 HxFuncInOut.c File Reference

```
#include "HxFuncInOut.h"
#include "HxCategories.h"
```

Row_variations

- template<class DataPtrT, class PixOpT> void **HxFuncInOut_Row_InTi** (DataPtrT ptr, int nPix, PixOpT &pixOp)

  Row : import, translation invariant.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut_Row_OutTi** (DataPtrT ptr, int nPix, PixOpT &pixOp)

  Row : export, translation invariant.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut_Row_InTv** (DataPtrT ptr, int nPix, PixOpT &pixOp, int x, int y, int z)

  Row : import, translation variant.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut_Row_OutTv** (DataPtrT ptr, int nPix, PixOpT &pixOp, int x, int y, int z)

  Row : export, translation variant.

InOut_variations

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpIn** dummy1, **HxTagTransInVar** dummy2, **HxTag1Phase** dummy3)

  Translation invariant, 1 phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpOut** dummy1, **HxTagTransInVar** dummy2, **HxTag1Phase** dummy3)

  Translation invariant, 1 phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpIn** dummy1, **HxTagTransVar** dummy2, **HxTag1Phase** dummy3)

  Translation variant, 1 phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpOut** dummy1, **HxTagTransVar** dummy2, **HxTag1Phase** dummy3)

  Translation variant, 1 phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpIn** dummy1, **HxTagTransInVar** dummy2, **HxTagNPhase** dummy3)

  Translation invariant, n phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpOut** dummy1, **HxTagTransInVar** dummy2, **HxTagNPhase** dummy3)

  Translation invariant, n phase pixel export operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpIn** dummy1, **HxTagTransVar** dummy2, **HxTagNPhase** dummy3)

  Translation variant, n phase pixel import operation.

- template<class DataPtrT, class PixOpT> void **HxFuncInOut** (DataPtrT ptr, **HxSizes** sizes, PixOpT &pixOp, **HxTagPixOpOut** dummy1, **HxTagTransVar** dummy2, **HxTagNPhase** dummy3)

  Translation variant, n phase pixel export operation.

Functions

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

  Dispatch function for HxFuncInOut (see **Global functions for InOut** (p. **??**)*).*

7.79.1 Detailed Description

7.79.2 Function Documentation

7.79.2.1 template<class DataPtrT, class PixOpT> void HxFuncInOut_Row_InTi (DataPtrT *ptr*, int *nPix*, PixOpT & *pixOp*) [inline]

Row : import, translation invariant.

```
63 {
64     while (--nPix >= 0)
65         ptr.writeIncX(pixOp.doIt());
66 }
```

7.79.2.2 template<class DataPtrT, class PixOpT> void HxFuncInOut_Row_OutTi (DataPtrT *ptr*, int *nPix*, PixOpT & *pixOp*) [inline]

Row : export, translation invariant.

```
72 {
73     while (--nPix >= 0)
74         pixOp.doIt(ptr.readIncX());
75 }
```

7.79.2.3 template<class DataPtrT, class PixOpT> void HxFuncInOut_Row_InTv (DataPtrT *ptr*, int *nPix*, PixOpT & *pixOp*, int *x*, int *y*, int *z*) [inline]

Row : import, translation variant.

```
81 {
82     while (--nPix >= 0)
83         ptr.writeIncX(pixOp.doIt(x++, y, z));
84 }
```

7.79.2.4 template<class DataPtrT, class PixOpT> void HxFuncInOut_Row_OutTv (DataPtrT *ptr*, int *nPix*, PixOpT & *pixOp*, int *x*, int *y*, int *z*) [inline]

Row : export, translation variant.

```
90 {
91     while (--nPix >= 0)
92         pixOp.doIt(ptr.readIncX(), x++, y, z);
93 }
```

7.79.2.5 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpIn *dummy1*, HxTagTransInVar *dummy2*, HxTag1Phase *dummy3*)

Translation invariant, 1 phase pixel import operation.

```
109 {
110     for (int z=0; z<sizes.z(); z++)
111     {
112         for (int y=0; y<sizes.y(); y++)
113         {
114             HxFuncInOut_Row_InTi(ptr, sizes.x(), pixOp);
115             ptr.incY();
116         }
117         ptr.decY(sizes.y());
118         ptr.incZ();
119     }
120 }
```

7.79.2.6 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpOut *dummy1*, HxTagTransInVar *dummy2*, HxTag1Phase *dummy3*)

Translation invariant, 1 phase pixel export operation.

```
129 {
130     for (int z=0; z<sizes.z(); z++)
131     {
132         for (int y=0; y<sizes.y(); y++)
133         {
134             HxFuncInOut_Row_OutTi(ptr, sizes.x(), pixOp);
135             ptr.incY();
136         }
```

```
137         ptr.decY(sizes.y());
138         ptr.incZ();
139     }
140 }
```

7.79.2.7 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpIn *dummy1*, HxTagTransVar *dummy2*, HxTag1Phase *dummy3*)

Translation variant, 1 phase pixel import operation.

```
149 {
150     for (int z=0; z<sizes.z(); z++)
151     {
152         for (int y=0; y<sizes.y(); y++)
153         {
154             HxFuncInOut_Row_InTv(ptr, sizes.x(), pixOp, 0, y, z);
155             ptr.incY();
156         }
157         ptr.decY(sizes.y());
158         ptr.incZ();
159     }
160 }
```

7.79.2.8 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpOut *dummy1*, HxTagTransVar *dummy2*, HxTag1Phase *dummy3*)

Translation variant, 1 phase pixel export operation.

```
169 {
170     for (int z=0; z<sizes.z(); z++)
171     {
172         for (int y=0; y<sizes.y(); y++)
173         {
174             HxFuncInOut_Row_OutTv(ptr, sizes.x(), pixOp, 0, y, z);
175             ptr.incY();
176         }
177         ptr.decY(sizes.y());
178         ptr.incZ();
179     }
180 }
```

7.79.2.9 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpIn *dummy1*, HxTagTransInVar *dummy2*, HxTagNPhase *dummy3*)

Translation invariant, n phase pixel import operation.

```
189 {
190     for (int phase=1; phase<=pixOp.nrPhases(); phase++) {
191         pixOp.init(phase);
192         for (int z=0; z<sizes.z(); z++)
193         {
```

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

7.79.2.10 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpOut *dummy1*, HxTagTransInVar *dummy2*, HxTagNPhase *dummy3*)

Translation invariant, n phase pixel export operation.

```
213 {
214     for (int phase=1; phase<=pixOp.nrPhases(); phase++) {
215         pixOp.init(phase);
216         for (int z=0; z<sizes.z(); z++)
217         {
218             for (int y=0; y<sizes.y(); y++)
219             {
220                 HxFuncInOut_Row_OutTi(ptr, sizes.x(), pixOp);
221                 ptr.incY();
222             }
223             ptr.decY(sizes.y());
224             ptr.incZ();
225         }
226         pixOp.done(phase);
227     }
228 }
```

7.79.2.11 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpIn *dummy1*, HxTagTransVar *dummy2*, HxTagNPhase *dummy3*)

Translation variant, n phase pixel import operation.

```
237 {
238     for (int phase=1; phase<=pixOp.nrPhases(); phase++) {
239         pixOp.init(phase);
240         for (int z=0; z<sizes.z(); z++)
241         {
242             for (int y=0; y<sizes.y(); y++)
243             {
244                 HxFuncInOut_Row_InTv(ptr, sizes.x(), pixOp, 0, y, z);
245                 ptr.incY();
246             }
247             ptr.decY(sizes.y());
248             ptr.incZ();
249         }
250         pixOp.done(phase);
251     }
252 }
```

7.79.2.12 template<class DataPtrT, class PixOpT> void HxFuncInOut (DataPtrT *ptr*, HxSizes *sizes*, PixOpT & *pixOp*, HxTagPixOpOut *dummy1*, HxTagTransVar *dummy2*, HxTagNPhase *dummy3*)

Translation variant, n phase pixel export operation.

```
261 {
262     for (int phase=1; phase<=pixOp.nrPhases(); phase++) {
263         pixOp.init(phase);
264         for (int z=0; z<sizes.z(); z++)
265         {
266             for (int y=0; y<sizes.y(); y++)
267             {
268                 HxFuncInOut_Row_OutTv(ptr, sizes.x(), pixOp, 0, y, z);
269                 ptr.incY();
270             }
271             ptr.decY(sizes.y());
272             ptr.incZ();
273         }
274         pixOp.done(phase);
275     }
276 }
```

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

Dispatch function for HxFuncInOut (see **Global functions for InOut** (p. **??**)).

Dispatch is based on the categories defined in PixOpT.

```
288 {
289     HxFuncInOut(
290         ptr, sizes, pixOp,
291         typename PixOpT::DirectionCategory(),
292         typename PixOpT::TransVarianceCategory(),
293         typename PixOpT::PhaseCategory());
294 }
```

7.80 HxFuncInOutInit.h File Reference

```
#include "HxCategories.h"
```

Functions

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes**, **HxTagList** &tags, const **HxTagPixOpOut** d1, const **HxTagTransInVar** d2, const **HxTag1Phase** d3)

  Initialization for translation invariant, 1 phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxTagPixOpOut** d1, const **HxTagTransVar** d2, const **HxTag1Phase** d3)

  Initialization for translation variant, 1 phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes**, **HxTagList** &tags, const **HxTagPixOpIn** d1, const **HxTagTransInVar** d2, const **HxTag1Phase** d3)

Initialization for translation invariant, 1 phase pixel import operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxTagPixOpIn** d1, const **HxTagTransVar** d2, const **HxTag1Phase** d3)

  Initialization for translation variant, 1 phase pixel import operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes**, **HxTagList** &tags, const **HxTagPixOpOut** d1, const **HxTagTransInVar** d2, const **HxTagNPhase** d3)

  Initialization for translation invariant, n phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxTagPixOpOut** d1, const **HxTagTransVar** d2, const **HxTagNPhase** d3)

  Initialization for translation variant, n phase pixel export operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes**, **HxTagList** &tags, const **HxTagPixOpIn** d1, const **HxTagTransInVar** d2, const **HxTagNPhase** d3)

  Initialization for translation invariant, n phase pixel import operation.

- template<class PixOpT> PixOpT **HxFuncInOutInit** (**HxSizes** sizes, **HxTagList** &tags, const **HxTagPixOpIn** d1, const **HxTagTransVar** d2, const **HxTagNPhase** d3)

  Initialization for translation variant, n phase pixel import operation.

7.80.1 Detailed Description

7.80.2 Function Documentation

7.80.2.1 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes, HxTagList & *tags*, const HxTagPixOpOut *d1*, const HxTagTransInVar *d2*, const HxTag1Phase *d3*) [inline]

Initialization for translation invariant, 1 phase pixel export operation.

```
25 {
26     PixOpT pixOp(tags);
27     return pixOp;
28 }
```

7.80.2.2 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes *sizes*, HxTagList & *tags*, const HxTagPixOpOut *d1*, const HxTagTransVar *d2*, const HxTag1Phase *d3*) [inline]

Initialization for translation variant, 1 phase pixel export operation.

```
37 {
38     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
39     return pixOp;
40 }
```

7.80.2.3 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes, HxTagList & *tags*, const HxTagPixOpIn *d1*, const HxTagTransInVar *d2*, const HxTag1Phase *d3*) [inline]

Initialization for translation invariant, 1 phase pixel import operation.

```
49 {
50     PixOpT pixOp(tags);
51     return pixOp;
52 }
```

7.80.2.4 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes *sizes*, HxTagList & *tags*, const HxTagPixOpIn *d1*, const HxTagTransVar *d2*, const HxTag1Phase *d3*) [inline]

Initialization for translation variant, 1 phase pixel import operation.

```
61 {
62     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
63     return pixOp;
64 }
```

7.80.2.5 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes, HxTagList & *tags*, const HxTagPixOpOut *d1*, const HxTagTransInVar *d2*, const HxTagNPhase *d3*) [inline]

Initialization for translation invariant, n phase pixel export operation.

```
73 {
74     PixOpT pixOp(tags);
75     return pixOp;
76 }
```

7.80.2.6 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes *sizes*, HxTagList & *tags*, const HxTagPixOpOut *d1*, const HxTagTransVar *d2*, const HxTagNPhase *d3*) [inline]

Initialization for translation variant, n phase pixel export operation.

```
85 {
86     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
87     return pixOp;
88 }
```

7.80.2.7 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes, HxTagList & *tags*, const HxTagPixOpIn *d1*, const HxTagTransInVar *d2*, const HxTagNPhase *d3*) [inline]

Initialization for translation invariant, n phase pixel import operation.

```
97 {
98     PixOpT pixOp(tags);
99     return pixOp;
100 }
```

7.80.2.8 template<class PixOpT> PixOpT HxFuncInOutInit (HxSizes *sizes*, HxTagList & *tags*, const HxTagPixOpIn *d1*, const HxTagTransVar *d2*, const HxTagNPhase *d3*) [inline]

Initialization for translation variant, n phase pixel import operation.

```
109 {
110     PixOpT pixOp(tags, sizes.x(), sizes.y(), sizes.z());
111     return pixOp;
112 }
```

7.81 HxFuncKernelNgbOp2d.c File Reference

```
#include "HxFuncKernelNgbOp2d.h"
#include "HxCategories.h"
```

Pix_variations

- template<class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d_Pix_P1Loop** (SrcDataPtrT srcPtr, NgbT &ngb, KernelT &kernel, int ngbWidth, int ngbHeight)

  Pix : phase 1, loop.

- template<class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d_Pix_P2Loop** (SrcDataPtrT srcPtr, NgbT &ngb, KernelT &kernel, int ngbWidth, int ngbHeight)

  Pix : phase 2, loop.

Row_variations

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, KernelT &kernel, **HxTag1Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 1, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, KernelT &kernel, **HxTag2Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 2, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, KernelT &kernel, **HxTagNPhase** dummy1, **HxTagLoop** dummy2)

  Row : phase N, loop.

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void **HxFuncKernelNgbOp2dDispatch** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb, KernelT &kernel)

Dispatch function for KernelNgbOp2d (see **Global functions for KernelNgbOp2d** (p. **??**)*).*

7.81.1 Detailed Description

7.81.2 Function Documentation

7.81.2.1 template<class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d_Pix_P1Loop (SrcDataPtrT *srcPtr*, NgbT & *ngb*, KernelT & *kernel*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 1, loop.

```
52 {
53     for (int j=0; j<ngbHeight; j++) {
54         for (int i=0; i<ngbWidth; i++) {
55             ngb.next(i, j, srcPtr.readIncX(), kernel(i, j));
56         }
57         srcPtr.decX(ngbWidth);
58         srcPtr.incY();
59     }
60 }
```

7.81.2.2 template<class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d_Pix_P2Loop (SrcDataPtrT *srcPtr*, NgbT & *ngb*, KernelT & *kernel*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 2, loop.

```
68 {
69     for (int j=0; j<ngbHeight; j++) {
70         for (int i=0; i<ngbWidth; i++) {
71             ngb.next2(i, j, srcPtr.readIncX(), kernel(i, j));
72         }
73         srcPtr.decX(ngbWidth);
74         srcPtr.incY();
75     }
76 }
```

7.81.2.3 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, KernelT & *kernel*, HxTag1Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 1, loop.

```
92 {
93     int ngbWidth = ngb.size().x();
94     int ngbHeight = ngb.size().y();
95     SrcDataPtrT cenSrcPtr = srcPtr;
96     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
97     for(int x=0; x < imgWidth; x++) {
98         ngb.init(x, y, cenSrcPtr.readIncX());
```

```
99          HxFuncKernelNgbOp2d_Pix_P1Loop(srcPtr, ngb, kernel, ngbWidth, ngbHeight);
100          srcPtr.incX();
101          dstPtr.writeIncX(ngb.result());
102      }
103 }
```

7.81.2.4 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, KernelT & *kernel*, HxTag2Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 2, loop.

```
112 {
113     int ngbWidth = ngb.size().x();
114     int ngbHeight = ngb.size().y();
115     SrcDataPtrT cenSrcPtr = srcPtr;
116     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
117     for(int x=0; x < imgWidth; x++) {
118         ngb.init(x, y, cenSrcPtr.read());
119         HxFuncKernelNgbOp2d_Pix_P1Loop(srcPtr, ngb, kernel, ngbWidth, ngbHeight);
120         ngb.init2(x, y, cenSrcPtr.readIncX());
121         HxFuncKernelNgbOp2d_Pix_P2Loop(srcPtr, ngb, kernel, ngbWidth, ngbHeight);
122         srcPtr.incX();
123         dstPtr.writeIncX(ngb.result());
124     }
125 }
```

7.81.2.5 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, KernelT & *kernel*, HxTagNPhase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase N, loop.

```
134 {
135     int ngbWidth = ngb.size().x();
136     int ngbHeight = ngb.size().y();
137     SrcDataPtrT cenSrcPtr = srcPtr;
138     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
139     for(int x=0; x < imgWidth; x++) {
140         int phase = 1;
141         do {
142             ngb.init(phase, x, y, cenSrcPtr.read());
143             HxFuncKernelNgbOp2d_Pix_P1Loop(srcPtr, ngb, kernel, ngbWidth,
144                                            ngbHeight);
145             ngb.done(phase);
146         } while (ngb.hasNextPhase(phase++));
147         cenSrcPtr.incX();
148         srcPtr.incX();
149         dstPtr.writeIncX(ngb.result());
150     }
151 }
```

7.81.2.6 template<class DstDataPtrT, class SrcDataPtrT, class NgbT, class KernelT> void HxFuncKernelNgbOp2dDispatch (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*, KernelT & *kernel*)

Dispatch function for KernelNgbOp2d (see **Global functions for KernelNgbOp2d** (p. **??**)).

Dispatch is based on the categories defined in NgbT.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, srcPtr is at (IX0,IY0)

kernel Input image, IS = kernelSize, IBS = 0

```
170 {
171     for(int y=0; y < dstSize.y(); y++) {
172         HxFuncKernelNgbOp2d_Row(dstPtr, srcPtr, y, dstSize.x(), ngb, kernel,
173                                 typename NgbT::PhaseCategory(),
174                                 typename NgbT::IteratorCategory());
175         srcPtr.incY();
176         dstPtr.incY();
177     }
178 }
```

7.82 HxFuncMNpo.c File Reference

```
#include "HxFuncMNpo.h"
#include "HxCategories.h"
```

Functions

- template<class DstDataPtrArray, class SrcDataPtrArray, class MNpoT> void **HxFuncMNpo** (DstDataPtrArray &dstPtrs, SrcDataPtrArray &srcPtrs, **HxSizes** dstSize, MNpoT &mpo, **HxTagTransInVar**)

  Translation invariant M output, N input pixel operation.

- template<class DstDataPtrArray, class SrcDataPtrArray, class MNpoT> void **HxFuncMNpoDispatch** (DstDataPtrArray &dstPtrs, SrcDataPtrArray &srcPtrs, **HxSizes** dstSize, MNpoT &mnpo)

  Dispatch function for M output, N input pixel operation.

7.82.1 Detailed Description

7.82.2 Function Documentation

7.82.2.1 template<class DstDataPtrArray, class SrcDataPtrArray, class MNpoT> void HxFuncMNpo (DstDataPtrArray & *dstPtrs*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, MNpoT & *mpo*, HxTagTransInVar)

Translation invariant M output, N input pixel operation.

```
26 {
27     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
28     int n = srcPtrs.size();
29     int m = dstPtrs.size();
30     typename SrcDataPtrArray::ArithType *src =
31     new typename SrcDataPtrArray::ArithType [n];
32     typename DstDataPtrArray::ArithType *dst =
33     new typename DstDataPtrArray::ArithType [m];
34
35     while (--nPix >= 0) {
36     int i;
37         for (i=0; i < n; i++)
38             src[i] = srcPtrs[i].readIncX();
39
40         mpo.doIt(dst, src);
41
42         for (i=0; i < m; i++)
43             dstPtrs[i].writeIncX(dst[i]);
44     }
45
46     delete [] dst;
47     delete [] src;
48 }
```

7.82.2.2 template<class DstDataPtrArray, class SrcDataPtrArray, class MNpoT> void HxFuncMNpoDispatch (DstDataPtrArray & *dstPtr*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, MNpoT & *mnpo*)

Dispatch function for M output, N input pixel operation.

Dispatch is based on the TransVarianceCategory category defined in MNpoT. Calls **HxFuncMNpo**(DstDataPtrArray&,SrcDataPtrArray&,HxSizes,MNpoT&,HxTagTransInVar) (p. 196) or **HxFuncMNpo**(DstDataPtrArray&,SrcDataPtrArray&,HxSizes,MNpoT&,HxTagTransInVar) (p. 196).

```
95 {
96     HxFuncMNpo(
97         dstPtrs, srcPtrs, dstSize, mnpo,
98         typename MNpoT::TransVarianceCategory());
99 }
```

7.83 HxFuncMpo.c File Reference

```
#include "HxFuncMpo.h"
#include "HxCategories.h"
```

Functions

- template<class DstDataPtrT, class SrcDataPtrArray, class MpoT> void **HxFuncMpo** (DstDataPtrT dstPtr, SrcDataPtrArray &srcPtrs, **HxSizes** dstSize, MpoT &mpo, **HxTagTransInVar**)

  Translation invariant multi pixel operation.

- template<class DstDataPtrT, class SrcDataPtrArray, class MpoT> void **HxFuncMpoDispatch** (DstDataPtrT dstPtr, SrcDataPtrArray &srcPtrs, **HxSizes** dstSize, MpoT &mpo)

  Dispatch function for multi pixel operation.

7.83.1 Detailed Description

7.83.2 Function Documentation

7.83.2.1 template<class DstDataPtrT, class SrcDataPtrArray, class MpoT> void HxFuncMpo (DstDataPtrT *dstPtr*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, MpoT & *mpo*, HxTagTransInVar)

Translation invariant multi pixel operation.

```
26 {
27     int n = srcPtrs.size();
28     typename SrcDataPtrArray::ArithType *src =
29         new typename SrcDataPtrArray::ArithType [n];
30
31     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
32     while (--nPix >= 0) {
33         for (int i=0; i < n; i++)
34             src[i] = srcPtrs[i].readIncX();
35         dstPtr.writeIncX(mpo.doIt(src));
36     }
37
38     delete [] src;
39 }
```

7.83.2.2 template<class DstDataPtrT, class SrcDataPtrArray, class MpoT> void HxFuncMpoDispatch (DstDataPtrT *dstPtr*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, MpoT & *mpo*)

Dispatch function for multi pixel operation.

Dispatch is based on the TransVarianceCategory category defined in MpoT. Calls **HxFuncMpo**(DstDataPtrT,SrcDataPtrArray&,HxSizes,MpoT&,HxTagTransInVar) (p. 198) or **HxFuncMpo**(DstDataPtrT,SrcDataPtrArray&,HxSizes,MpoT&,HxTagTransVar) (p. 198).

```
77 {
78     HxFuncMpo(
79         dstPtr, srcPtrs, dstSize, mpo,
80         typename MpoT::TransVarianceCategory());
81 }
```

7.84 HxFuncNgbOp2d.c File Reference

```
#include "HxFuncNgbOp2d.h"
#include "HxCategories.h"
```

Pix_variations

- template<class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Pix_P1Cnum** (SrcDataPtrT &srcPtr, NgbT &ngb, typename NgbT::CnumType cnum)

  Pix : phase 1, cnum.

- template<class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Pix_P1Loop** (SrcDataPtrT srcPtr, NgbT &ngb, int ngbWidth, int ngbHeight)

  Pix : phase 1, loop.

- template<class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Pix_P2Loop** (SrcDataPtrT srcPtr, NgbT &ngb, int ngbWidth, int ngbHeight)

  Pix : phase 2, loop.

Row_variations

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, **HxTag1Phase** dummy1, **HxTagCnum** dummy2)

  Row : phase 1, cnum.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, **HxTag1Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 1, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, **HxTag2Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 2, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2d_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int y, int imgWidth, NgbT &ngb, **HxTagNPhase** dummy1, **HxTagLoop** dummy2)

  Row : phase N, loop.

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void **HxFuncNgbOp2dDispatch** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, NgbT &ngb)

  Dispatch function for NgbOp2d (see **Global functions for NgbOp2d** (p. **??**)*) Dispatch is based on the categories defined in NgbT.*

7.84.1 Detailed Description

7.84.2 Function Documentation

7.84.2.1 template<class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Pix_P1Cnum (SrcDataPtrT & *srcPtr*, NgbT & *ngb*, typename NgbT::CnumType *cnum*) [inline]

Pix : phase 1, cnum.

```
54 {
55     for (; cnum != ngb.end(); cnum.inc())
56     {
57         SrcDataPtrT nPtr(srcPtr);
58         nPtr.incXYZ(cnum.x(), cnum.y());
59         ngb.next(cnum.x(), cnum.y(), nPtr.read());
60     }
61 }
```

7.84.2.2 template<class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Pix_P1Loop (SrcDataPtrT *srcPtr*, NgbT & *ngb*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 1, loop.

```
68 {
69     for (int j=0; j<ngbHeight; j++) {
70         for (int i=0; i<ngbWidth; i++) {
71             ngb.next(i, j, srcPtr.readIncX());
72         }
73         srcPtr.decX(ngbWidth);
74         srcPtr.incY();
75     }
76 }
```

7.84.2.3 template<class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Pix_P2Loop (SrcDataPtrT *srcPtr*, NgbT & *ngb*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 2, loop.

```
83 {
84     for (int j=0; j<ngbHeight; j++) {
85         for (int i=0; i<ngbWidth; i++) {
86             ngb.next2(i, j, srcPtr.readIncX());
87         }
88         srcPtr.decX(ngbWidth);
89         srcPtr.incY();
90     }
91 }
```

7.84.2.4 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag1Phase *dummy1*, HxTagCnum *dummy2*) [inline]

Row : phase 1, cnum.

```
106 {
107     srcPtr.incXYZ(ngb.size().x()/2, ngb.size().y()/2, 0);
108     for(int x=0; x < imgWidth; x++) {
109         ngb.init(x, y, srcPtr.read());
110         HxFuncNgbOp2d_Pix_P1Cnum(srcPtr, ngb, ngb.begin());
111         srcPtr.incX();
112         dstPtr.writeIncX(ngb.result());
113     }
114 }
```

7.84.2.5 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag1Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 1, loop.

```
122 {
123     int ngbWidth = ngb.size().x();
124     int ngbHeight = ngb.size().y();
125     SrcDataPtrT cenSrcPtr = srcPtr;
126     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
127     for(int x=0; x < imgWidth; x++) {
128         ngb.init(x, y, cenSrcPtr.readIncX());
129         HxFuncNgbOp2d_Pix_P1Loop(srcPtr, ngb, ngbWidth, ngbHeight);
130         srcPtr.incX();
131         dstPtr.writeIncX(ngb.result());
132     }
133 }
```

7.84.2.6 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag2Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 2, loop.

```
141 {
142     int ngbWidth = ngb.size().x();
143     int ngbHeight = ngb.size().y();
144     SrcDataPtrT cenSrcPtr = srcPtr;
145     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
146     for(int x=0; x < imgWidth; x++) {
147         ngb.init(x, y, cenSrcPtr.read());
148         HxFuncNgbOp2d_Pix_P1Loop(srcPtr, ngb, ngbWidth, ngbHeight);
149         ngb.init2(x, y, cenSrcPtr.readIncX());
150         HxFuncNgbOp2d_Pix_P2Loop(srcPtr, ngb, ngbWidth, ngbHeight);
151         srcPtr.incX();
152         dstPtr.writeIncX(ngb.result());
153     }
154 }
```

7.84.2.7 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2d_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTagNPhase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase N, loop.

```
162 {
163     int ngbWidth = ngb.size().x();
164     int ngbHeight = ngb.size().y();
165     SrcDataPtrT cenSrcPtr = srcPtr;
166     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
167     for(int x=0; x < imgWidth; x++) {
168         int phase = 1;
169         do {
170             ngb.init(phase, x, y, cenSrcPtr.read());
171             HxFuncNgbOp2d_Pix_P1Loop(srcPtr, ngb, ngbWidth, ngbHeight);
```

```
172             ngb.done(phase);
173         } while (ngb.hasNextPhase(phase++));
174         cenSrcPtr.incX();
175         srcPtr.incX();
176         dstPtr.writeIncX(ngb.result());
177     }
178 }
```

7.84.2.8 template<class DstDataPtrT, class SrcDataPtrT, class NgbT> void HxFuncNgbOp2dDispatch (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, NgbT & *ngb*)

Dispatch function for NgbOp2d (see **Global functions for NgbOp2d** (p. **??**)) Dispatch is based on the categories defined in NgbT.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, srcPtr is at (IX0,IY0)

```
196 {
197     for(int y=0; y < dstSize.y(); y++) {
198         HxFuncNgbOp2d_Row(dstPtr, srcPtr, y, dstSize.x(), ngb,
199                           typename NgbT::PhaseCategory(),
200                           typename NgbT::IteratorCategory());
201         srcPtr.incY();
202         dstPtr.incY();
203     }
204 }
```

7.85 HxFuncNgbOp2dExtra.c File Reference

```
#include "HxFuncNgbOp2dExtra.h"
#include "HxCategories.h"
```

Pix_variations

- template<class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Pix_P1Cnum** (SrcDataPtrT &srcPtr, ExtraDataPtrT &extraPtr, NgbT &ngb, typename NgbT::CnumType cnum)

  Pix : phase 1, cnum.

- template<class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Pix_P1Loop** (SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, NgbT &ngb, int ngbWidth, int ngbHeight)

  Pix : phase 1, loop.

- template<class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Pix_P2Loop** (SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, NgbT &ngb, int ngbWidth, int ngbHeight)

  Pix : phase 2, loop.

Row_variations

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, int y, int imgWidth, NgbT &ngb, **HxTag1Phase** dummy1, **HxTagCnum** dummy2)

  Row : phase 1, cnum.

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, int y, int imgWidth, NgbT &ngb, **HxTag1Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 1, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, int y, int imgWidth, NgbT &ngb, **HxTag2Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 2, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtra_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, int y, int imgWidth, NgbT &ngb, **HxTagNPhase** dummy1, **HxTagLoop** dummy2)

  Row : phase N, loop.

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void **HxFuncNgbOp2dExtraDispatch** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, **HxSizes** dstSize, NgbT &ngb)

  Dispatch function for NgbOp2dExtra (see **Global functions for NgbOp2dExtra** (p. **??**)*) Dispatch is based on the categories defined in NgbT.*

7.85.1 Detailed Description

7.85.2 Function Documentation

7.85.2.1 template<class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Pix_P1Cnum (SrcDataPtrT & *srcPtr*, ExtraDataPtrT & *extraPtr*, NgbT & *ngb*, typename NgbT::CnumType *cnum*) [inline]

Pix : phase 1, cnum.

```
54 {
55     for (; cnum != ngb.end(); cnum.inc())
56     {
57         SrcDataPtrT nPtr(srcPtr);
58         nPtr.incXYZ(cnum.x(), cnum.y());
59         ExtraDataPtrT n2Ptr(extraPtr);
60         n2Ptr.incXYZ(cnum.x(), cnum.y());
61         ngb.next(cnum.x(), cnum.y(), nPtr.read(), n2Ptr.read());
62     }
63 }
```

7.85.2.2 template<class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Pix_P1Loop (SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, NgbT & *ngb*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 1, loop.

```
71 {
72     for (int j=0; j<ngbHeight; j++) {
73         for (int i=0; i<ngbWidth; i++) {
74             ngb.next(i, j, srcPtr.readIncX(), extraPtr.readIncX());
75         }
76         srcPtr.decX(ngbWidth);
77         srcPtr.incY();
78         extraPtr.decX(ngbWidth);
79         extraPtr.incY();
80     }
81 }
```

7.85.2.3 template<class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Pix_P2Loop (SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, NgbT & *ngb*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 2, loop.

```
89 {
90     for (int j=0; j<ngbHeight; j++) {
91         for (int i=0; i<ngbWidth; i++) {
92             ngb.next2(i, j, srcPtr.readIncX(), extraPtr.readIncX());
93         }
94         srcPtr.decX(ngbWidth);
95         srcPtr.incY();
96         extraPtr.decX(ngbWidth);
97         extraPtr.incY();
98     }
99 }
```

7.85.2.4 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag1Phase *dummy1*, HxTagCnum *dummy2*) [inline]

Row : phase 1, cnum.

```
115 {
116     srcPtr.incXYZ(ngb.size().x()/2, ngb.size().y()/2, 0);
117     extraPtr.incXYZ(ngb.size().x()/2, ngb.size().y()/2, 0);
118     for(int x=0; x < imgWidth; x++) {
119         ngb.init(x, y, srcPtr.read(), extraPtr.read());
120         HxFuncNgbOp2dExtra_Pix_P1Cnum(srcPtr, extraPtr, ngb, ngb.begin());
121         srcPtr.incX();
122         extraPtr.incX();
123         dstPtr.writeIncX(ngb.result());
124     }
125 }
```

7.85.2.5 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag1Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 1, loop.

```
134 {
135     int ngbWidth = ngb.size().x();
136     int ngbHeight = ngb.size().y();
137     SrcDataPtrT cenSrcPtr = srcPtr;
138     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
139     ExtraDataPtrT cenExtraPtr = extraPtr;
140     cenExtraPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
141     for(int x=0; x < imgWidth; x++) {
142         ngb.init(x, y, cenSrcPtr.readIncX(), cenExtraPtr.readIncX());
143         HxFuncNgbOp2dExtra_Pix_P1Loop(srcPtr, extraPtr, ngb, ngbWidth, ngbHeight);
144         srcPtr.incX();
145         extraPtr.incX();
146         dstPtr.writeIncX(ngb.result());
147     }
148 }
```

7.85.2.6 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag2Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 2, loop.

```
157 {
158     int ngbWidth = ngb.size().x();
159     int ngbHeight = ngb.size().y();
160     SrcDataPtrT cenSrcPtr = srcPtr;
161     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
162     ExtraDataPtrT cenExtraPtr = extraPtr;
163     cenExtraPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
164     for(int x=0; x < imgWidth; x++) {
165         ngb.init(x, y, cenSrcPtr.read(), cenExtraPtr.read());
166         HxFuncNgbOp2dExtra_Pix_P1Loop(srcPtr, extraPtr, ngb, ngbWidth, ngbHeight);
167         ngb.init2(x, y, cenSrcPtr.readIncX(), cenExtraPtr.readIncX());
168         HxFuncNgbOp2dExtra_Pix_P2Loop(srcPtr, extraPtr, ngb, ngbWidth, ngbHeight);
169         srcPtr.incX();
170         extraPtr.incX();
171         dstPtr.writeIncX(ngb.result());
172     }
173 }
```

7.85.2.7 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtra_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTagNPhase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase N, loop.

```
182 {
```

```
183     int ngbWidth = ngb.size().x();
184     int ngbHeight = ngb.size().y();
185     SrcDataPtrT cenSrcPtr = srcPtr;
186     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
187     ExtraDataPtrT cenExtraPtr = extraPtr;
188     cenExtraPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
189     for(int x=0; x < imgWidth; x++) {
190         int phase = 1;
191         do {
192             ngb.init(phase, x, y, cenSrcPtr.read(), cenExtraPtr.read());
193             HxFuncNgbOp2dExtra_Pix_P1Loop(srcPtr, extraPtr, ngb,
194                                           ngbWidth, ngbHeight);
195             ngb.done(phase);
196         } while (ngb.hasNextPhase(phase++));
197         cenSrcPtr.incX();
198         srcPtr.incX();
199         cenExtraPtr.incX();
200         extraPtr.incX();
201         dstPtr.writeIncX(ngb.result());
202     }
203 }
```

7.85.2.8 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class NgbT> void HxFuncNgbOp2dExtraDispatch (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, HxSizes *dstSize*, NgbT & *ngb*)

Dispatch function for NgbOp2dExtra (see **Global functions for NgbOp2dExtra** (p. **??**)) Dispatch is based on the categories defined in NgbT.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, srcPtr is at (IX0,IY0)

extraPtr Extra image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, extraPtr is at (IX0,IY0)

```
223 {
224     for(int y=0; y < dstSize.y(); y++) {
225         HxFuncNgbOp2dExtra_Row(dstPtr, srcPtr, extraPtr, y, dstSize.x(), ngb,
226                                typename NgbT::PhaseCategory(),
227                                typename NgbT::IteratorCategory());
228         srcPtr.incY();
229         extraPtr.incY();
230         dstPtr.incY();
231     }
232 }
```

7.86 HxFuncNgbOp2dExtra2.c File Reference

```
#include "HxFuncNgbOp2dExtra2.h"
#include "HxCategories.h"
```

Pix_variations

- template<class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Pix_P1Cnum** (SrcDataPtrT &srcPtr, ExtraDataPtrT &extraPtr, Extra2DataPtrT &extra2Ptr, NgbT &ngb, typename NgbT::CnumType cnum)

Pix : phase 1, cnum.

- template<class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Pix_P1Loop** (SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, NgbT &ngb, int ngbWidth, int ngbHeight)

  Pix : phase 1, loop.

- template<class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Pix_P2Loop** (SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, NgbT &ngb, int ngbWidth, int ngbHeight)

  Pix : phase 2, loop.

Row_variations

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, int y, int imgWidth, NgbT &ngb, **HxTag1Phase** dummy1, **HxTagCnum** dummy2)

  Row : phase 1, cnum.

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, int y, int imgWidth, NgbT &ngb, **HxTag1Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 1, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, int y, int imgWidth, NgbT &ngb, **HxTag2Phase** dummy1, **HxTagLoop** dummy2)

  Row : phase 2, loop.

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, int y, int imgWidth, NgbT &ngb, **HxTagNPhase** dummy1, **HxTagLoop** dummy2)

  Row : phase N, loop.

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void **HxFuncNgbOp2dExtra2Dispatch** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, ExtraDataPtrT extraPtr, Extra2DataPtrT extra2Ptr, **HxSizes** dstSize, NgbT &ngb)

  Dispatch function for NgbOp2dExtra2 (see **Global functions for NgbOp2dExtra2** (p. **??**)*) Dispatch is based on the categories defined in NgbT.*

7.86.1 Detailed Description

7.86.2 Function Documentation

7.86.2.1 template<class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Pix_P1Cnum (SrcDataPtrT & *srcPtr*, ExtraDataPtrT & *extraPtr*, Extra2DataPtrT & *extra2Ptr*, NgbT & *ngb*, typename NgbT::CnumType *cnum*) [inline]

Pix : phase 1, cnum.

```
55 {
56     for (; cnum != ngb.end(); cnum.inc())
57     {
58         SrcDataPtrT nPtr(srcPtr);
59         nPtr.incXYZ(cnum.x(), cnum.y());
60         ExtraDataPtrT n2Ptr(extraPtr);
61         n2Ptr.incXYZ(cnum.x(), cnum.y());
62         Extra2DataPtrT n3Ptr(extra2Ptr);
63         n3Ptr.incXYZ(cnum.x(), cnum.y());
64         ngb.next(cnum.x(), cnum.y(), nPtr.read(), n2Ptr.read(), n3Ptr.read());
65     }
66 }
```

7.86.2.2 template<class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Pix_P1Loop (SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, NgbT & *ngb*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 1, loop.

```
75 {
76     for (int j=0; j<ngbHeight; j++) {
77         for (int i=0; i<ngbWidth; i++) {
78             ngb.next(i, j, srcPtr.readIncX(), extraPtr.readIncX(),
79                      extra2Ptr.readIncX());
80         }
81         srcPtr.decX(ngbWidth);
82         srcPtr.incY();
83         extraPtr.decX(ngbWidth);
84         extraPtr.incY();
85         extra2Ptr.decX(ngbWidth);
86         extra2Ptr.incY();
87     }
88 }
```

7.86.2.3 template<class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Pix_P2Loop (SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, NgbT & *ngb*, int *ngbWidth*, int *ngbHeight*) [inline]

Pix : phase 2, loop.

```
97 {
98     for (int j=0; j<ngbHeight; j++) {
99         for (int i=0; i<ngbWidth; i++) {
100             ngb.next2(i, j, srcPtr.readIncX(), extraPtr.readIncX(),
```

```
101                     extra2Ptr.readIncX());
102         }
103         srcPtr.decX(ngbWidth);
104         srcPtr.incY();
105         extraPtr.decX(ngbWidth);
106         extraPtr.incY();
107         extra2Ptr.decX(ngbWidth);
108         extra2Ptr.incY();
109     }
110 }
```

7.86.2.4 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag1Phase *dummy1*, HxTagCnum *dummy2*) [inline]

Row : phase 1, cnum.

```
127 {
128     srcPtr.incXYZ(ngb.size().x()/2, ngb.size().y()/2, 0);
129     extraPtr.incXYZ(ngb.size().x()/2, ngb.size().y()/2, 0);
130     extra2Ptr.incXYZ(ngb.size().x()/2, ngb.size().y()/2, 0);
131     for(int x=0; x < imgWidth; x++) {
132         ngb.init(x, y, srcPtr.read(), extraPtr.read(), extra2Ptr.read());
133         HxFuncNgbOp2dExtra2_Pix_P1Cnum(srcPtr, extraPtr, extra2Ptr, ngb,
134                                        ngb.begin());
135         srcPtr.incX();
136         extraPtr.incX();
137         extra2Ptr.incX();
138         dstPtr.writeIncX(ngb.result());
139     }
140 }
```

7.86.2.5 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag1Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 1, loop.

```
150 {
151     int ngbWidth = ngb.size().x();
152     int ngbHeight = ngb.size().y();
153     SrcDataPtrT cenSrcPtr = srcPtr;
154     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
155     ExtraDataPtrT cenExtraPtr = extraPtr;
156     cenExtraPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
157     Extra2DataPtrT cenExtra2Ptr = extra2Ptr;
158     cenExtra2Ptr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
159     for(int x=0; x < imgWidth; x++) {
160         ngb.init(x, y, cenSrcPtr.readIncX(), cenExtraPtr.readIncX(),
161                  cenExtra2Ptr.readIncX());
162         HxFuncNgbOp2dExtra2_Pix_P1Loop(srcPtr, extraPtr, extra2Ptr, ngb,
163                                        ngbWidth, ngbHeight);
164         srcPtr.incX();
165         extraPtr.incX();
166         extra2Ptr.incX();
```

```
167         dstPtr.writeIncX(ngb.result());
168     }
169 }
```

7.86.2.6 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTag2Phase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase 2, loop.

```
179 {
180     int ngbWidth = ngb.size().x();
181     int ngbHeight = ngb.size().y();
182     SrcDataPtrT cenSrcPtr = srcPtr;
183     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
184     ExtraDataPtrT cenExtraPtr = extraPtr;
185     cenExtraPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
186     Extra2DataPtrT cenExtra2Ptr = extra2Ptr;
187     cenExtra2Ptr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
188     for(int x=0; x < imgWidth; x++) {
189         ngb.init(x, y, cenSrcPtr.read(), cenExtraPtr.read(), cenExtra2Ptr.read());
190         HxFuncNgbOp2dExtra2_Pix_P1Loop(srcPtr, extraPtr, extra2Ptr, ngb,
191                                        ngbWidth, ngbHeight);
192         ngb.init2(x, y, cenSrcPtr.readIncX(), cenExtraPtr.readIncX(),
193                   cenExtraPtr.readIncX());
194         HxFuncNgbOp2dExtra2_Pix_P2Loop(srcPtr, extraPtr, extra2Ptr, ngb,
195                                        ngbWidth, ngbHeight);
196         srcPtr.incX();
197         extraPtr.incX();
198         extra2Ptr.incX();
199         dstPtr.writeIncX(ngb.result());
200     }
201 }
```

7.86.2.7 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, int *y*, int *imgWidth*, NgbT & *ngb*, HxTagNPhase *dummy1*, HxTagLoop *dummy2*) [inline]

Row : phase N, loop.

```
211 {
212     int ngbWidth = ngb.size().x();
213     int ngbHeight = ngb.size().y();
214     SrcDataPtrT cenSrcPtr = srcPtr;
215     cenSrcPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
216     ExtraDataPtrT cenExtraPtr = extraPtr;
217     cenExtraPtr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
218     Extra2DataPtrT cenExtra2Ptr = extra2Ptr;
219     cenExtra2Ptr.incXYZ(ngbWidth/2, ngbHeight/2, 0);
220     for(int x=0; x < imgWidth; x++) {
221         int phase = 1;
222         do {
223             ngb.init(phase, x, y, cenSrcPtr.read(), cenExtraPtr.read(),
224                      cenExtra2Ptr.read());
225             HxFuncNgbOp2dExtra2_Pix_P1Loop(srcPtr, extraPtr, extra2Ptr, ngb,
```

```
226                                         ngbWidth, ngbHeight);
227             ngb.done(phase);
228         } while (ngb.hasNextPhase(phase++));
229         cenSrcPtr.incX();
230         srcPtr.incX();
231         cenExtraPtr.incX();
232         extraPtr.incX();
233         cenExtra2Ptr.incX();
234         extra2Ptr.incX();
235         dstPtr.writeIncX(ngb.result());
236     }
237 }
```

7.86.2.8 template<class DstDataPtrT, class SrcDataPtrT, class ExtraDataPtrT, class Extra2DataPtrT, class NgbT> void HxFuncNgbOp2dExtra2Dispatch (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, ExtraDataPtrT *extraPtr*, Extra2DataPtrT *extra2Ptr*, HxSizes *dstSize*, NgbT & *ngb*)

Dispatch function for NgbOp2dExtra2 (see **Global functions for NgbOp2dExtra2** (p. **??**)) Dispatch is based on the categories defined in NgbT.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, srcPtr is at (IX0,IY0)

extraPtr Extra image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, extraPtr is at (IX0,IY0)

extra2Ptr Extra image: IS = dstSize+2∗(ngbSize/2), IBS = ngbSize/2, extra2Ptr is at (IX0,IY0)

```
260 {
261     for(int y=0; y < dstSize.y(); y++) {
262         HxFuncNgbOp2dExtra2_Row(dstPtr, srcPtr, extraPtr, extra2Ptr,
263                                 y, dstSize.x(), ngb,
264                                 typename NgbT::PhaseCategory(),
265                                 typename NgbT::IteratorCategory());
266         srcPtr.incY();
267         extraPtr.incY();
268         extra2Ptr.incY();
269         dstPtr.incY();
270     }
271 }
```

7.87 HxFuncRecGenConv2d.c File Reference

```
#include "HxFuncRecGenConv2d.h"
#include "HxEnvironment.h"
```

Functions

- template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void **HxFuncRecGenConv2d_XYdirSim** (DataPtrType imgPtr, KerDataPtrType kerPtr, ArithType dummy, **HxSizes** imgSize, **HxSizes** kerSize, PixOpT &pixOp, RedOpT &redOp)

  RecGenConv2d : XY direction, simple.

- template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dDispatch** (DataPtrType imgPtr, KerDataPtrType kerPtr, ArithType dummy, **HxSizes** imgSize, **HxSizes** kerSize, PixOpT &pixOp, RedOpT &redOp)

  Dispatch function for RecGenConv2d.

7.87.1 Detailed Description

7.87.2 Function Documentation

7.87.2.1 template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void HxFuncRecGenConv2d_XYdirSim (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, ArithType *dummy*, HxSizes *imgSize*, HxSizes *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*)

RecGenConv2d : XY direction, simple.

```
26 {
27     int kerWidth = kerSize.x();
28     int kerHeight = kerSize.y();
29     int x, y, i, j;
30     int borderWidth = kerWidth/2;
31     int borderHeight = kerHeight/2;
32     imgSize = imgSize - HxSizes(2*borderWidth, 2*borderHeight, 0);
33     int imgWidth = imgSize.x();
34     int imgHeight = imgSize.y();
35
36     ArithType result, tmpVal, neutralElement(RedOpT::neutralElement());
37
38     // scan order
39
40     DataPtrType iPtr = imgPtr;
41     for (y=0 ; y<imgHeight ; y++) {
42         for (x=0 ; x<imgWidth ; x++) {
43             DataPtrType nPtr = iPtr;
44             KerDataPtrType kPtr = kerPtr;
45             result = neutralElement;
46             for (j=0 ; j<borderHeight ; j++) {
47                 for (i=0 ; i<kerWidth ; i++) {
48                     tmpVal = pixOp.doIt(nPtr.readIncX(), kPtr.readIncX());
49                     redOp.doIt(result, tmpVal);
50                 }
51                 nPtr.decX(kerWidth);
52                 nPtr.incY();
53             }
54             for (i=0 ; i<=borderWidth ; i++) {
55                 tmpVal = pixOp.doIt(nPtr.readIncX(), kPtr.readIncX());
56                 redOp.doIt(result, tmpVal);
57             }
58             nPtr.decX();
59             nPtr.write(result);
60             iPtr.incX();
61         }
62         iPtr.decX(imgWidth);
63         iPtr.incY();
64     }
65
66     // anti scan order
67
68     iPtr = imgPtr;
```

```
69     iPtr.incXYZ(borderWidth+imgWidth+borderWidth-1,
70                 borderHeight+imgHeight+borderHeight-1, 0);
71     for (y=0 ; y<imgHeight ; y++) {
72         for (x=0 ; x<imgWidth ; x++) {
73             DataPtrType nPtr = iPtr;
74             KerDataPtrType kPtr = kerPtr;
75             kPtr.incX(kerWidth-1);
76             kPtr.incY(kerHeight-1);
77             result = neutralElement;
78             for (j=0 ; j<borderHeight ; j++) {
79                 for (i=0 ; i<kerWidth ; i++) {
80                     tmpVal = pixOp.doIt(nPtr.read(), kPtr.read());
81                     redOp.doIt(result, tmpVal);
82                     nPtr.decX(); kPtr.decX();
83                 }
84                 nPtr.incX(kerWidth);
85                 nPtr.decY();
86             }
87             for (i=0 ; i<=borderWidth ; i++) {
88                 tmpVal = pixOp.doIt(nPtr.read(), kPtr.read());
89                 redOp.doIt(result, tmpVal);
90                 nPtr.decX(); kPtr.decX();
91             }
92             nPtr.incX();
93             nPtr.write(result);
94             iPtr.decX();
95         }
96         iPtr.incX(imgWidth);
97         iPtr.decY();
98     }
99 }
```

7.87.2.2 template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void HxFuncRecGenConv2dDispatch (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, ArithType *dummy*, HxSizes *imgSize*, HxSizes *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*)

Dispatch function for RecGenConv2d.

Assertions:

Parameters:

imgPtr Input/Output image: IS = imgSize, IBS = kerSize/2, imgPtr is at (IX0,IY0)

kerPtr Input image, IS = kerSize, IBS = 0

```
115 {
116     HxFuncRecGenConv2d_XYdirSim(imgPtr, kerPtr, dummy, imgSize,
117                                 kerSize, pixOp, redOp);
118 }
```

7.88 HxFuncRecGenConv2dK1d.c File Reference

```
#include "HxFuncRecGenConv2dK1d.h"
#include "HxEnvironment.h"
```

Line_variations

- template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_Line_XdirSim** (DataPtrT imgPtr, int imgWidth, ArithT *kernel, int kerSize, PixOpT &pixOp, RedOpT &redOp)

  Line : X direction, simple.

- template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_Line_YdirSim** (DataPtrT imgPtr, int imgHeight, ArithT *kernel, int kerSize, ArithT *ngbBuf, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, simple.

- template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_Line_YdirBuf** (DataPtrT imgPtr, int imgWidth, ArithT *kernel, int kerSize, PixOpT &pixOp, RedOpT &redOp)

  Line : Y direction, buffered.

- template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_Line_YdirBufAnti** (DataPtrT imgPtr, int imgWidth, ArithT *kernel, int kerSize, PixOpT &pixOp, RedOpT &redOp)

RecGenConv2dK1d_variations

- template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_XdirSim** (DataPtrType imgPtr, KerDataPtrType kerPtr, ArithType dummy, **HxSizes** imgSize, **HxSizes** borderSize, **HxSizes** kerSize, PixOpT &pixOp, RedOpT &redOp)

  RecGenConv2dK1d : X direction, simple.

- template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_YdirSim** (DataPtrType imgPtr, KerDataPtrType kerPtr, ArithType dummy, **HxSizes** imgSize, **HxSizes** borderSize, **HxSizes** kerSize, PixOpT &pixOp, RedOpT &redOp)

  RecGenConv2dK1d : Y direction, simple.

- template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1d_YdirBuf** (DataPtrType imgPtr, KerDataPtrType kerPtr, ArithType dummy, **HxSizes** imgSize, **HxSizes** borderSize, **HxSizes** kerSize, PixOpT &pixOp, RedOpT &redOp)

  RecGenConv2dK1d : Y direction, buffered.

Functions

- template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void **HxFuncRecGenConv2dK1dDispatch** (DataPtrType imgPtr, KerDataPtrType kerPtr, ArithType dummy, **HxSizes** imgSize, **HxSizes** borderSize, **HxSizes** kerSize, PixOpT &pixOp, RedOpT &redOp, int dimension, bool buffered)

  Dispatch function for RecGenConv2dK1d (see **Global functions for RecGenConv2dK1d** (p. **??**)*) Dispatch is based on dimension and inplace parameters.*

7.88.1 Detailed Description

7.88.2 Function Documentation

7.88.2.1 template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1d_Line_XdirSim (DataPtrT *imgPtr*, int *imgWidth*, ArithT * *kernel*, int *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : X direction, simple.

```
52 {
53     int x, n, ngbSize = kerSize/2, scanSize = imgWidth+ngbSize;
54     ArithT result;
55
56     // scan order
57
58     for (x=0; x<scanSize; x++)
59     {
60         result = pixOp.doIt(imgPtr.readIncX(), kernel[0]);
61         for (n=1; n<=ngbSize; n++)
62             redOp.doIt(result, pixOp.doIt(imgPtr.readIncX(), kernel[n]));
63         imgPtr.decX();
64         imgPtr.write(result);
65         imgPtr.decX(ngbSize-1);
66     }
67
68     // anti scan order
69
70     imgPtr.decX();
71     kernel += kerSize/2;
72
73     for (x=0; x<imgWidth; x++)
74     {
75         result = pixOp.doIt(imgPtr.readIncX(), kernel[0]);
76         for (n=1; n<=ngbSize; n++)
77             redOp.doIt(result, pixOp.doIt(imgPtr.readIncX(), kernel[n]));
78         imgPtr.decX(ngbSize+1);
79         imgPtr.write(result);
80         imgPtr.decX();
81     }
82
83 }
```

7.88.2.2 template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1d_Line_YdirSim (DataPtrT *imgPtr*, int *imgHeight*, ArithT * *kernel*, int *kerSize*, ArithT * *ngbBuf*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, simple.

```
92 {
93     int y, n, ngbSize = kerSize/2, scanSize = imgHeight+ngbSize;
94     ArithT result;
95     ArithT* ngbPtr = ngbBuf;
96
97     // scan order
98
99     for (n=0; n<ngbSize; n++)
100     {
101         ngbBuf[n] = imgPtr.read();
```

```
102         imgPtr.incY();
103     }
104
105
106     for (y=0; y<scanSize; y++)
107     {
108         ngbPtr[ngbSize] = imgPtr.read();
109         imgPtr.incY();
110         result = pixOp.doIt(ngbPtr[0], kernel[0]);
111         for (n=1; n<=ngbSize; n++)
112             redOp.doIt(result, pixOp.doIt(ngbPtr[n], kernel[n]));
113         ngbPtr[ngbSize] = result;
114         ngbPtr++;
115     }
116
117     // anti scan order
118
119     imgPtr.decY(ngbSize+1);
120     ngbPtr -= 1;
121     kernel += ngbSize;
122
123     for (y=0; y<imgHeight; y++)
124     {
125         result = pixOp.doIt(ngbPtr[0], kernel[0]);
126         for (n=1; n<=ngbSize; n++)
127             redOp.doIt(result, pixOp.doIt(ngbPtr[n], kernel[n]));
128         imgPtr.write(*ngbPtr-- = result);
129         imgPtr.decY();
130     }
131 }
```

7.88.2.3 template<class DataPtrT, class ArithT, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1d_Line_YdirBuf (DataPtrT *imgPtr*, int *imgWidth*, ArithT * *kernel*, int *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*) [static]

Line : Y direction, buffered.

```
139 {
140     const int ngbSize = kerSize/2;
141     int x, n;
142     ArithT result;
143     DataPtrT linePtr(imgPtr);
144
145     // scan order
146
147     for (x=0; x<imgWidth; x++)
148         imgPtr.writeIncX(pixOp.doIt(imgPtr.read(), kernel[ngbSize]));
149
150     imgPtr.decX(imgWidth);
151
152     // previous lines
153     for (n=ngbSize-1; n>=0; n--)
154     {
155         linePtr.decY();
156         for (x=0; x<imgWidth; x++) {
157             result = imgPtr.read();
158             redOp.doIt(result, pixOp.doIt(linePtr.readIncX(), kernel[n]));
159             imgPtr.writeIncX(result);
160         }
161         imgPtr.decX(imgWidth);
162         linePtr.decX(imgWidth);
```

```
163     }
164 }
```

7.88.2.4 template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1d_XdirSim (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, ArithType *dummy*, HxSizes *imgSize*, HxSizes *borderSize*, HxSizes *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*)

RecGenConv2dK1d : X direction, simple.

```
216 {
217     int borderWidth = borderSize.x() > 0 ? borderSize.x() : kerSize.x()/2;
218     int borderHeight = borderSize.y();
219     imgSize = imgSize - HxSizes(2*borderWidth, 2*borderHeight, 0);
220     int imgWidth = imgSize.x();
221     int imgHeight = imgSize.y();
222     int y, i;
223 
224     HxPixelAllocator<ArithType> allocator;
225 
226     ArithType* kernel = allocator.allocate(kerSize.x());
227 
228     for (i=0; i<kerSize.x(); i++)
229         kernel[i] = kerPtr.readIncX();
230 
231     imgPtr.incXYZ(borderWidth-kerSize.x()/2, borderHeight, 0);
232 
233     for (y=0; y<imgHeight; y++)
234     {
235         HxFuncRecGenConv2dK1d_Line_XdirSim(
236             imgPtr, imgWidth, kernel, kerSize.x(), pixOp, redOp);
237         imgPtr.incY();
238     }
239 
240     allocator.deallocate(kernel);
241 }
```

7.88.2.5 template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1d_YdirSim (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, ArithType *dummy*, HxSizes *imgSize*, HxSizes *borderSize*, HxSizes *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*)

RecGenConv2dK1d : Y direction, simple.

```
251 {
252     int borderWidth = borderSize.x();
253     int borderHeight = borderSize.y() > 0 ? borderSize.y() : kerSize.x()/2;
254     imgSize = imgSize - HxSizes(2*borderWidth, 2*borderHeight, 0);
255     int imgWidth = imgSize.x();
256     int imgHeight = imgSize.y();
257     int x, i;
258 
259     HxPixelAllocator<ArithType> allocator;
260 
261     ArithType* kernel = allocator.allocate(kerSize.x());
262 
263     for (i=0; i<kerSize.x(); i++)
```

```
264         kernel[i] = kerPtr.readIncX();
265
266     ArithType* ngbBuf = allocator.allocate(imgHeight+kerSize.x());
267
268     imgPtr.incXYZ(borderWidth, borderHeight-kerSize.x()/2, 0);
269
270     for (x=0; x<imgWidth; x++)
271     {
272         HxFuncRecGenConv2dK1d_Line_YdirSim(
273             imgPtr, imgHeight, kernel, kerSize.x(), ngbBuf, pixOp, redOp);
274         imgPtr.incX();
275     }
276
277     allocator.deallocate(ngbBuf);
278     allocator.deallocate(kernel);
279 }
```

7.88.2.6 template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1d_YdirBuf (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, ArithType *dummy*, HxSizes *imgSize*, HxSizes *borderSize*, HxSizes *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*)

RecGenConv2dK1d : Y direction, buffered.

```
290 {
291     HxEnvironment::instance()->outputStream() << "Buffered" << STD_ENDL;
292     int borderWidth = borderSize.x();
293     int borderHeight = borderSize.y() > 0 ? borderSize.y() : kerSize.x()/2;
294     imgSize = imgSize - HxSizes(2*borderWidth, 2*borderHeight, 0);
295     int imgWidth = imgSize.x();
296     int imgHeight = imgSize.y();
297     int y, i;
298
299     HxPixelAllocator<ArithType> allocator;
300
301     ArithType* kernel = allocator.allocate(kerSize.x());
302
303     for (i=0; i<kerSize.x(); i++)
304         kernel[i] = kerPtr.readIncX();
305
306     imgPtr.incXYZ(borderWidth, borderHeight, 0);
307
308     for (y=0; y<imgHeight+kerSize.x()/2; y++)
309     {
310         HxFuncRecGenConv2dK1d_Line_YdirBuf(
311             imgPtr, imgWidth, kernel, kerSize.x(), pixOp, redOp);
312         imgPtr.incY();
313     }
314     imgPtr.decY(kerSize.x()/2);
315
316     for (y=imgHeight-1; y>=0; y--)
317     {
318         imgPtr.decY();
319         HxFuncRecGenConv2dK1d_Line_YdirBufAnti(
320             imgPtr, imgWidth, kernel, kerSize.x(), pixOp, redOp);
321     }
322
323     allocator.deallocate(kernel);
324 }
```

7.88.2.7 template<class DataPtrType, class KerDataPtrType, class ArithType, class PixOpT, class RedOpT> void HxFuncRecGenConv2dK1dDispatch (DataPtrType *imgPtr*, KerDataPtrType *kerPtr*, ArithType *dummy*, HxSizes *imgSize*, HxSizes *borderSize*, HxSizes *kerSize*, PixOpT & *pixOp*, RedOpT & *redOp*, int *dimension*, bool *buffered*)

Dispatch function for RecGenConv2dK1d (see **Global functions for RecGenConv2dK1d** (p. **??**)) Dispatch is based on dimension and inplace parameters.

Assertions:

Parameters:

imgPtr Input/Output image: IS = imgSize, IBS = borderSize, imgPtr is at (IX0,IY0)

kerPtr Input image, IS = kerSize, IBS = 0

```
345 {
346     switch (dimension) {
347     case 1  :
348         HxFuncRecGenConv2dK1d_XdirSim(imgPtr, kerPtr, dummy, imgSize, borderSize,
349                                 kerSize, pixOp, redOp);
350         break;
351     case 2  :
352         if (buffered)
353             HxFuncRecGenConv2dK1d_YdirBuf(imgPtr, kerPtr, dummy, imgSize, borderSize,
354                                     kerSize, pixOp, redOp);
355         else
356             HxFuncRecGenConv2dK1d_YdirSim(imgPtr, kerPtr, dummy, imgSize, borderSize,
357                                     kerSize, pixOp, redOp);
358         break;
359     default :
360         HxEnvironment::instance()->errorStream()
361             << "HxFuncRecGenConv2dK1dDispatch: "
362             << "cannot execute convolution in dimension " << dimension
363             << STD_ENDL;
364         HxEnvironment::instance()->flush();
365     }
366 }
```

7.89 HxFuncRgbOp2d.c File Reference

```
#include "HxFuncRgbOp2d.h"
#include "HxCategories.h"
#include "HxFuncs2d.h"
```

Functions

- template<class DataPtrT, class RgbT> void **HxFuncRgbOp2d** (DataPtrT imPtr, **HxSizes** imSize, int *pixels, int resWidth, int resHeight, **HxGeoIntType** gi, RgbT &rgb)

  Function for RgbOp2d.

7.89.1 Detailed Description

7.89.2 Function Documentation

7.89.2.1 template<class DataPtrT, class RgbT> void HxFuncRgbOp2d (DataPtrT *imPtr*, HxSizes *imSize*, int * *pixels*, int *resWidth*, int *resHeight*, HxGeoIntType *gi*, RgbT & *rgb*)

Function for RgbOp2d.

```
25 {
26     int width = imSize.x();
27     int height = imSize.y();
28
29
30     if (((resWidth == -1) && (resHeight == -1)) ||
31                 ((resWidth == width) && (resHeight == height))) {
32         int nPix = width * height;
33         DataPtrT p = imPtr;
34         while (--nPix >= 0) {
35             *pixels++ = rgb.doIt(p.read());
36             p.incX();
37         }
38         return;
39     }
40
41     double sx = (double) width / resWidth;
42     double sy = (double) height / resHeight;
43     if (gi == NEAREST) {
44         for (int y=0 ; y<resHeight ; y++) {
45             for (int x=0 ; x<resWidth ; x++) {
46                 DataPtrT p = imPtr;
47                 p.incXYZ((int) (sx * x + 0.5), (int) (sy * y + 0.5), 0);
48                 *pixels++ = rgb.doIt(p.read());
49             }
50         }
51         return;
52     }
53
54     typedef typename RgbT::ArithTypeDouble ArithTypeDouble;
55     ArithTypeDouble result;
56     for (int y=0 ; y<resHeight ; y++) {
57         for (int x=0 ; x<resWidth ; x++) {
58             result = HxFunc2dSample(imPtr, result, sx * x, sy * y, 0, gi);
59             *pixels++ = rgb.doItDouble(result);
60         }
61     }
62 }
```

7.90 HxFuncRgbOp3d.c File Reference

```
#include "HxFuncRgbOp3d.h"
#include "HxCategories.h"
#include "HxFuncs3d.h"
```

Functions

- template<class DataPtrT, class RgbT> void **HxFuncRgbOp3d** (DataPtrT imPtr, **HxSizes** imSize, int *pixels, int dimension, int coordinate, int resWidth, int resHeight, **HxGeoIntType** gi, RgbT &rgb)

  Function for RgbOp3d.

7.90.1 Detailed Description

7.90.2 Function Documentation

7.90.2.1 template<class DataPtrT, class RgbT> void HxFuncRgbOp3d (DataPtrT *imPtr*, HxSizes *imSize*, int * *pixels*, int *dimension*, int *coordinate*, int *resWidth*, int *resHeight*, HxGeoIntType *gi*, RgbT & *rgb*)

Function for RgbOp3d.

```
26 {
27     int x, y, z;
28     HxSizes planeSize;
29     switch (dimension) {
30     case 1:
31         planeSize = HxSizes(imSize.y(), imSize.z(), 1);
32         break;
33     case 2:
34         planeSize = HxSizes(imSize.x(), imSize.z(), 1);
35         break;
36     case 3:
37         planeSize = HxSizes(imSize.x(), imSize.y(), 1);
38         break;
39     }
40
41     if (((resWidth == -1) && (resHeight == -1)) ||
42             ((resWidth == planeSize.x()) && (resHeight == planeSize.y()))) {
43         DataPtrT p = imPtr;
44         switch (dimension) {
45         case 1:
46             for (z=0 ; z<imSize.z() ; z++) {
47                 p = imPtr;
48                 p.incXYZ(coordinate, 0, z);
49                 for (y=0 ; y<imSize.y() ; y++) {
50                     *pixels++ = rgb.doIt(p.read());
51                     p.incY();
52                 }
53             }
54             break;
55         case 2:
56             for (z=0 ; z<imSize.z() ; z++) {
57                 p = imPtr;
58                 p.incXYZ(0, coordinate, z);
59                 for (x=0 ; x<imSize.x() ; x++) {
60                     *pixels++ = rgb.doIt(p.read());
61                     p.incX();
62                 }
63             }
64             break;
65         case 3:
66             imPtr.incZ(coordinate);
67             int nPix = imSize.x() * imSize.y();
```

```
68              while (--nPix >= 0) {
69                  *pixels++ = rgb.doIt(imPtr.read());
70                  imPtr.incX();
71              }
72              break;
73          }
74          return;
75      }
76
77      double sx = (double) planeSize.x() / resWidth;
78      double sy = (double) planeSize.y() / resHeight;
79      typedef typename RgbT::ArithTypeDouble ArithTypeDouble;
80      ArithTypeDouble result;
81      switch (dimension) {
82      case 1:
83          for (y=0 ; y<resHeight ; y++) {
84              for (x=0 ; x<resWidth ; x++) {
85                  result = HxFunc3dSample(imPtr, result, coordinate, sx * x, sy * y, gi);
86                  *pixels++ = rgb.doItDouble(result);
87              }
88          }
89          break;
90      case 2:
91          for (y=0 ; y<resHeight ; y++) {
92              for (x=0 ; x<resWidth ; x++) {
93                  result = HxFunc3dSample(imPtr, result, sx * x, coordinate, sy * y, gi);
94                  *pixels++ = rgb.doItDouble(result);
95              }
96          }
97          break;
98      case 3:
99          for (y=0 ; y<resHeight ; y++) {
100              for (x=0 ; x<resWidth ; x++) {
101                  result = HxFunc3dSample(imPtr, result, sx * x, sy * y, coordinate, gi);
102                  *pixels++ = rgb.doItDouble(result);
103              }
104          }
105          break;
106      }
107 }
```

7.91 HxFuncSet.c File Reference

```
#include "HxFuncSet.h"
#include "HxCategories.h"
#include <memory.h>
```

Typedefs

- typedef **HxDataPtr2dScalarTem**< double, **HxScalarDouble** > **DataPtr2dDouble**
- typedef **HxDataPtr2dTem**< **HxVec2Double**, **HxVec2Double** > **DataPtr2dVec2Double**

Functions

- template<class DstDataPtrT, class SrcDataPtrT> void **HxFuncSet_Row** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, int nPix)

Set one line of nPix from src to dst.

- template<> void **HxFuncSet_Row< DataPtr2dDouble, DataPtr2dDouble >** (DataPtr2dDouble dPtr, DataPtr2dDouble sPtr, int nPix)

  Set one line of nPix from src to dst, template specialization for 2d double images.

- template<> void **HxFuncSet_Row< DataPtr2dVec2Double, DataPtr2dVec2Double >** (DataPtr2dVec2Double dPtr, DataPtr2dVec2Double sPtr, int nPix)

  Set one line of nPix from src to dst, template specialization for 2d Vec2Double images.

- template<class DstDataPtrT, class SrcDataPtrT> void **HxFuncSet** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** regionSize)

  Set function.

7.91.1 Detailed Description

7.91.2 Function Documentation

7.91.2.1 template<class DstDataPtrT, class SrcDataPtrT> void HxFuncSet_Row (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, int *nPix*)

Set one line of nPix from src to dst.

```
24 {
25     while (--nPix >= 0)
26         dstPtr.writeIncX(srcPtr.readIncX());
27 }
```

7.91.2.2 template<> void HxFuncSet_Row< DataPtr2dDouble, DataPtr2dDouble > (DataPtr2dDouble *dPtr*, DataPtr2dDouble *sPtr*, int *nPix*) [inline]

Set one line of nPix from src to dst, template specialization for 2d double images.

7.91.2.3 template<> void HxFuncSet_Row< DataPtr2dVec2Double, DataPtr2dVec2Double > (DataPtr2dVec2Double *dPtr*, DataPtr2dVec2Double *sPtr*, int *nPix*) [inline]

Set one line of nPix from src to dst, template specialization for 2d Vec2Double images.

7.91.2.4 template<class DstDataPtrT, class SrcDataPtrT> void HxFuncSet (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *regionSize*)

Set function.

Sets pixels designated by dstPtr equal to pixels designated by srcPtr. regionSize defines the 3D region of pixels involved.

Note that the function does NOT assume the pixels in the region to be contiguous in memory. However, a row of pixels must be contiguous.

```
69 {
70     int y, z;
71
72     int lineSize = regionSize.x();
73
74     for (z = 0; z < regionSize.z(); z++) {
75         for (y = 0; y < regionSize.y(); y++) {
76             SrcDataPtrT sPtr(srcPtr);
77             DstDataPtrT dPtr(dstPtr);
78             sPtr.incXYZ(0, y, z);
79             dPtr.incXYZ(0, y, z);
80             HxFuncSet_Row(dPtr, sPtr, lineSize);
81         }
82     }
83 }
```

7.92 HxFuncUpo.c File Reference

```
#include "HxFuncUpo.h"
#include "HxCategories.h"
```

Functions

- template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void **HxFuncUpo** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, UpoT &upo, **HxTagTransInVar** dummy)

  Translation invariant unary pixel operation.

- template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void **HxFuncUpo** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, UpoT &upo, **HxTagTransVar** dummy)

  Translation variant unary pixel operation.

- template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void **HxFuncUpoDispatch** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, UpoT &upo)

  Dispatch function for unary pixel operation.

7.92.1 Detailed Description

7.92.2 Function Documentation

7.92.2.1 template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void HxFuncUpo (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, UpoT & *upo*, HxTagTransInVar *dummy*)

Translation invariant unary pixel operation.

```
26 {
27     int nPix = dstSize.x() * dstSize.y() * dstSize.z();
28     while (--nPix >= 0)
29         dstPtr.writeIncX(upo.doIt(srcPtr.readIncX()));
30 }
```

7.92.2.2 template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void HxFuncUpo (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, UpoT & *upo*, HxTagTransVar *dummy*)

Translation variant unary pixel operation.

```
39 {
40     for (int z=0 ; z<dstSize.z() ; z++) {
41         for (int y=0 ; y<dstSize.y() ; y++) {
42             for (int x=0 ; x<dstSize.x() ; x++) {
43                 dstPtr.writeIncX(upo.doIt(srcPtr.readIncX(), x, y, z));
44             }
45         }
46     }
47 }
```

7.92.2.3 template<class DstDataPtrT, class SrcDataPtrT, class UpoT> void HxFuncUpoDispatch (DstDataPtrT *dstPtr*, SrcDataPtrT *srcPtr*, HxSizes *dstSize*, UpoT & *upo*)

Dispatch function for unary pixel operation.

Dispatch is based on the TransVarianceCategory category defined in UpoT. Calls **HxFuncUpo**(DstDataPtrT,SrcDataPtrT,HxSizes,UpoT&,HxTagTransInVar) (p. 224) or **HxFuncUpo**(DstDataPtrT,SrcDataPtrT,HxSizes,UpoT&,HxTagTransVar) (p. 225).

```
60 {
61     HxFuncUpo(
62         dstPtr, srcPtr, dstSize, upo,
63         typename UpoT::TransVarianceCategory());
64 }
```

7.93 HxGauss.h File Reference

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

Functions

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

  Convolution Gaussian.

7.93.1 Detailed Description

7.93.2 Function Documentation

7.93.2.1 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     HxString fname("HxGauss");
20
21     if (img.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (sigma <= 0.0)
28     {
29         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of sigma", HxGlobalError::HX_GE_INVA
30         return HxImageRep();
31     }
32     if (truncation < 0.0)
33     {
34         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of truncation", HxGlobalError::HX_GE
35         return HxImageRep();
36     }
37
38
39     int minSize = HxImageMinSize(img);
40     HxImageRep gauss = HxMakeGaussian1d(sigma, 0, truncation, minSize);
41     return img.genConvSeparated(
42         gauss, "mul", "addAssign", HxImageRep::ARITH_PREC);
43 }
```

7.94 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.

7.94.1 Detailed Description

7.94.2 Function Documentation

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

Convolution Gaussian.

Equivalent to : img.genConv2dSep(gaussx, gaussy, ”mul”, ”addAssign”, HxImageRep::ARITH_PREC)

where gaussx and gaussy are the 1d double-precision Gaussian kernel based on sigma, orderDeriv{x,y}, 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     HxString fname("HxGaussDerivative2d");
20
21     if (img.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (sigma <= 0.0)
28     {
29         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of sigma", HxGlobalError::HX_GE_INVA
30         return HxImageRep();
31     }
32     if (orderDerivx < 0)
33     {
34         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of OrderDerivx", HxGlobalError::HX_G
35         return HxImageRep();
36     }
37     if (orderDerivy < 0)
38     {
39         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of OrderDerivy", HxGlobalError::HX_G
40         return HxImageRep();
41     }
42
43     if (truncation < 0.0)
44     {
45         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of truncation", HxGlobalError::HX_GE
46         return HxImageRep();
47     }
48
49     if (img.dimensionality() != 2)
50     {
51         HxGlobalError::instance()->reportError(fname, img.name(), "function only valid for 2D images", HxGlobalError
52         return HxImageRep();
53     }
54
55     HxImageRep gaussx, gaussy;
56
57     gaussx = HxMakeGaussian1d(sigma, orderDerivx, truncation,
58         img.dimensionSize(1));
59     gaussy = HxMakeGaussian1d(sigma, orderDerivy, truncation,
60         img.dimensionSize(2));
61
62     return img.genConv2dSep(
63         gaussx, gaussy, "mul", "addAssign", HxImageRep::ARITH_PREC);
64 }
```

7.95 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.

7.95.1 Detailed Description

7.95.2 Function Documentation

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

Convolution Gaussian.

Equivalent to : img.genConv3dSep(gaussx, gaussy, gaussx, ”mul”, ”addAssign”, HxImageRep::ARITH_PREC)

where gauss{x,y,z} is the 1d double-precision Gaussian kernel based on sigma, orderDeriv{x,y,z}, 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     HxString fname("HxGaussDerivative3d");
20 
21     if (img.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (sigma <= 0.0)
28     {
29         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of sigma", HxGlobalError::HX_GE_INVA
30         return HxImageRep();
31     }
32     if (orderDerivx < 0)
33     {
34         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of OrderDerivx", HxGlobalError::HX_
35         return HxImageRep();
36     }
37     if (orderDerivy < 0)
38     {
39         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of OrderDerivy", HxGlobalError::HX_
40         return HxImageRep();
41     }
42     if (orderDerivz < 0)
43     {
44         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of OrderDerivz", HxGlobalError::HX_
45         return HxImageRep();
46     }
47 
48     if (truncation < 0.0)
49     {
50         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of truncation", HxGlobalError::HX_GI
51         return HxImageRep();
52     }
53 
54     if (img.dimensionality() != 3)
55     {
56         HxGlobalError::instance()->reportError(fname, img.name(), "function only valid for 3D images", HxGlobalError
57         return HxImageRep();
```

```
58     }
59
60     HxImageRep gaussx, gaussy, gaussz;
61
62     gaussx = HxMakeGaussian1d(sigma, orderDerivx, truncation,
63         img.dimensionSize(1));
64     gaussy = HxMakeGaussian1d(sigma, orderDerivy, truncation,
65         img.dimensionSize(2));
66     gaussz = HxMakeGaussian1d(sigma, orderDerivz, truncation,
67         img.dimensionSize(3));
68
69     return img.genConv3dSep(gaussx, gaussy, gaussz,
70                             "mul", "addAssign", HxImageRep::ARITH_PREC);
71 }
```

7.96 HxGaussianDeblur.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxGaussianDeblur** (**HxImageRep** im, double dr=0.5, double dc=0.5)

  To reduce the variance of the Gausian blur by dr in the row direction and by dc in in the column direction, apply the mask 0 -0.5dr 0 -0.5dc 1+dr+dc -0.5dc 0 -0.5dr 0.

7.96.1 Detailed Description

7.96.2 Function Documentation

7.96.2.1 HxImageRep L_HXIMAGEREP HxGaussianDeblur (HxImageRep *im*, double *dr* = 0.5, double *dc* = 0.5)

To reduce the variance of the Gausian blur by dr in the row direction and by dc in in the column direction, apply the mask 0 -0.5dr 0 -0.5dc 1+dr+dc -0.5dc 0 -0.5dr 0.

Reference: John Immerkær. ”Use of Blur Space for Deblurring and Edge Preserving Noise Smoothing”, in IEEE Transactions on Image Processing, 2000.

```
15 {
16     int pixelDimensionality=1;
17     int dimensions=2;
18     HxSizes sizes(3,3,1);
19     double data[9]={    0,  -0.5*dr,    0,
20                     -0.5*dc,1+dr+dc, -0.5*dc,
21                         0,  -0.5*dr,    0};
22
23     HxImageRep kernel = HxMakeFromDoubleData(
24                     pixelDimensionality,
25                     dimensions,
26                     sizes,
27                     data);
28
29     return im.generalizedConvolution(
```

```
30     kernel, "mul", "addAssign", HxImageRep::ARITH_PREC);
31 }
```

7.97 HxGeodesicDistanceTransform.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxGeodesicDistanceTransform** (**HxImageRep** input, int conn=4)

  using the Luc Vincent watershed.

7.97.1 Detailed Description

7.97.2 Function Documentation

7.97.2.1 HxImageRep L_HXIMAGEREP HxGeodesicDistanceTransform (HxImageRep *input*, int *conn* = 4)

using the Luc Vincent watershed.

```
29 {
30     HxImageRep mask=input;
31     HxTagList tags;
32     HxAddTag(tags, "connectivity", conn);
33 
34     //to speedup compute here the min and max gray values
35     int     hmin,hmax;
36     hmin = HxPixMin(input).HxScalarIntValue().x();
37     hmax = HxPixMax(input).HxScalarIntValue().x();
38     HxAddTag(tags,"hmin",hmin);
39     HxAddTag(tags,"hmax",hmax);
40 
41     Array2D<int> *distImage = new Array2D<int>(input.sizes().x(),input.sizes().y(),0);
42     HxAddTag(tags,"geoDistPointer",distImage);
43 
44 
45     HxAddTag(tags,"exportGeodesicImage",1);
46     HxImageRep out = input.queueBasedOp(mask, "qWaterShedLV", tags);
47 
48     distImage = HxGetTag(tags,"geoDistPointer",distImage);
49     out = distImage->MakeHxImage();
50 
51     return out;
52 }
```

7.98 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)

7.98.1 Detailed Description

7.98.2 Enumeration Type Documentation

7.98.2.1 enum HxGeoIntType

Enumeration of geometric interpolation types.

Currently: NEAREST, LINEAR

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

7.99 HxGetBlobFeatures.h File Reference

```
#include "HxVec3Byte.h"
#include <vector>
```

Compounds

- struct **_basicFeatures**

Typedefs

- typedef _basicFeatures **BasicFeatures**
- typedef vector< BasicFeatures * > **BasicFeaturesList**

Functions

- BasicFeaturesList L_HXIMAGEREP **HxGetBlobFeatures** (**HxImageRep** input, **HxImageRep** &output, int conn, int minArea)

7.99.1 Detailed Description

7.100 HxGetPoints.h File Reference

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

Functions

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

7.100.1 Detailed Description

7.101 HxGetValues.h File Reference

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

Functions

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

7.101.1 Detailed Description

7.102 HxGreaterEqual.h File Reference

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

Functions

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

  Greater equal.

7.102.1 Detailed Description

7.102.2 Function Documentation

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

Greater equal.

The function performs greater equal (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoGreaterEqual** (p. 431). The image functor instantiator : **HxInstantiatorGreaterEqual** (p. 882).

```
13 {
14     HxString fname("HxGreaterEqual");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if ((im1.pixelDimensionality() != 1) || (im2.pixelDimensionality() != 1))
33     {
34         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
35         return HxImageRep();
36     }
37 
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.dimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
54         }
55     }
56 
57 
58     return im1.binaryPixOp(im2, "greaterEqual");
59 }
```

7.103 HxGreaterEqualVal.h File Reference

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

Functions

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

  Greater equal.

7.103.1 Detailed Description

7.103.2 Function Documentation

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

Greater equal.

The function performs greater equal (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoGreaterEqual** (p. 431). The image functor instantiator : **HxInstantiatorGreaterEqualV** (p. 883).

```
13 {
14     HxString fname("HxGreaterEqualVal");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
24         return HxImageRep();
25     }
26     if ((val.tag() != HxValue::SI) && (val.tag() != HxValue::SD))
27     {
28         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
29         return HxImageRep();
30     }
31 
32     return im.binaryPixOp(val, "greaterEqual");
33 }
```

7.104 HxGreaterThan.h File Reference

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

Functions

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

  Greater than.

7.104.1 Detailed Description

7.104.2 Function Documentation

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

Greater than.

The function performs greater than (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoGreaterThan** (p. 432). The image functor instantiator : **HxInstantiatorGreaterThan** (p. 883).

```
13 {
14     HxString fname("HxGreaterThan");
15
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if ((im1.pixelDimensionality() != 1) || (im2.pixelDimensionality() != 1))
33     {
34         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
35         return HxImageRep();
36     }
37
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.dimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
54         }
55     }
56
57     return im1.binaryPixOp(im2, "greaterThan");
58 }
```

7.105 HxGreaterThanVal.h File Reference

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

Functions

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

Greater than.

7.105.1 Detailed Description

7.105.2 Function Documentation

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

Greater than.

The function performs greater than (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoGreaterThan** (p. 432). The image functor instantiator : **HxInstantiatorGreaterThanV** (p. 884).

```
13 {
14     HxString fname("HxGreaterThanVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
24         return HxImageRep();
25     }
26     if ((val.tag() != HxValue::SI) && (val.tag() != HxValue::SD))
27     {
28         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
29         return HxImageRep();
30     }
31
32     return im.binaryPixOp(val, "greaterThan");
33 }
```

7.106 HxHighlightRegion.h File Reference

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

Functions

- **HxImageRep HxHighlightRegion** (**HxImageRep** im, **HxImageRep** mask, int label, double factor)

  HighlightRegionition.

7.106.1 Detailed Description

7.106.2 Function Documentation

7.106.2.1 HxImageRep HxHighlightRegion (HxImageRep *im*, HxImageRep *mask*, int *label*, double *factor*)

HighlightRegionition.

```
122 {
123     HxTagList tags;
124     HxAddTag(tags, "label", label);
125     HxAddTag(tags, "factor", factor);
126 
127     return im.binaryPixOp(mask, "highlightRegion", tags);
128 }
```

7.107 HxHilditchSkeleton.h File Reference

Functions

- **HxImageRep** L_HXIMAGEREP **HxHilditchSkeleton** (**HxImageRep** im)

  Hilditch skeleton.

7.107.1 Detailed Description

7.107.2 Function Documentation

7.107.2.1 HxImageRep L_HXIMAGEREP HxHilditchSkeleton (HxImageRep *im*)

Hilditch skeleton.

```
19 {
20     HxImageRep res=im;
21     bool changed= false;
22     HxTagList tags;
23     HxAddTag(tags,"changed",false);
24     int nriter=0;
25     do{
26         changed= false;
27         res = res.neighbourhoodOp("hilditch", tags);
28         changed = HxGetTag<bool>(tags,"changed");
29 //      printf("iteration# %d\n", nriter);
30         nriter++;
31     }while(changed && nriter<250);
32     //  printf("nr iterations %d\n", nriter);
33 
34 
35     return res;
36 }
```

7.108 HxHitOrMiss.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxHitOrMiss** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)
  sf1 and sf2 have to be disjoint!

7.108.1 Detailed Description

7.108.2 Function Documentation

7.108.2.1 HxImageRep L_HXIMAGEREP HxHitOrMiss (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

sf1 and sf2 have to be disjoint!

```
16 {
17     HxImageRep imc = HxSubVal(HxComplement(im),254);
18
19     return HxAnd(HxErosion(im,sf1), HxErosion(imc,sf2));
20 }
```

7.109 HxIdentMaskCentralMoments.h File Reference

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

Functions

- **HxValueList** L_HXIMAGEREP **HxIdentMaskCentralMoments** (**HxImageRep** im, **HxImage-Rep** mask, **HxPoint** p, **HxSizes** size, int label, int order)
  Compute the central moments of all pixels in "im" identified by "mask".

7.109.1 Detailed Description

7.109.2 Function Documentation

7.109.2.1 HxValueList L_HXIMAGEREP HxIdentMaskCentralMoments (HxImageRep *im*, HxImageRep *mask*, HxPoint *p*, HxSizes *size*, int *label*, int *order*)

Compute the central moments of all pixels in "im" identified by "mask".

"mask" is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point "p" with given "size" and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)

- Variation : **Translation variant, N phase export operation with an extra image** (p. 40)
- The pixel functor : **HxExportExtraIdentMaskCentralMoments** (p. 530)
- Instantiations : **HxInstExportExtraIdentMaskCentralMoments_c** (p. **??**)

```
15 {
16     HxValueList valList;
17     HxBoundingBox bb(size);
18     bb = bb.translate(p);
19     HxTagList tags;
20     HxAddTag(tags, "maskVal", label);
21     HxAddTag(tags, "boundingBox", bb);
22     HxAddTag(tags, "order", order);
23     HxAddTag(tags, "valList", &valList);
24     im.exportOpExtra("identMaskCentralMoments", mask, tags);
25     return valList;
26 }
```

7.110 HxIdentMaskMean.h File Reference

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

Functions

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

  Compute the mean of all pixels in "im" identified by "mask".

7.110.1 Detailed Description

7.110.2 Function Documentation

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

Compute the mean of all pixels in "im" identified by "mask".

"mask" is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point "p" with given "size" and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- The pixel functor : **HxInstExportExtraIdentMaskMean** (p. 930)
- Instantiations : **HxInstExportExtraIdentMaskMean_c** (p. **??**)

```
14 {
15     HxBoundingBox bb(size);
16     bb = bb.translate(p);
17     HxTagList tags;
18     HxAddTag(tags, "maskVal", label);
19     HxAddTag(tags, "boundingBox", bb);
```

```
20     im.exportOpExtra("identMaskMean", mask, tags);
21     HxValue v = HxGetTag(tags, "result", HxValue(0));
22     return v;
23 }
```

7.111 HxIdentMaskMedian.h File Reference

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

Functions

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

  Compute the median of all pixels in "im" identified by "mask".

7.111.1 Detailed Description

7.111.2 Function Documentation

7.111.2.1 HxValue L_HXIMAGEREP HxIdentMaskMedian (HxImageRep *im*, HxImageRep *mask*, HxPoint *p*, HxSizes *size*, int *label*)

Compute the median of all pixels in "im" identified by "mask".

"mask" is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point "p" with given "size" and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- The pixel functor : **HxInstExportExtraIdentMaskMedian** (p. 931)
- Instantiations : **HxInstExportExtraIdentMaskMedian_c** (p. **??**)

```
15 {
16     HxBoundingBox bb(size);
17     bb = bb.translate(p);
18     HxTagList tags;
19     HxAddTag(tags, "maskVal", label);
20     HxAddTag(tags, "boundingBox", bb);
21     im.exportOpExtra("identMaskMedian", mask, tags);
22     HxValue v = HxGetTag(tags, "result", HxValue(0));
23     return v;
24 }
```

7.112 HxIdentMaskMoments.h File Reference

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

Functions

- **HxValueList** L_HXIMAGEREP **HxIdentMaskMoments** (**HxImageRep** im, **HxImageRep** mask, **HxPoint** p, **HxSizes** size, int label, int order)

  Compute the moments of all pixels in "im" identified by "mask".

7.112.1 Detailed Description

7.112.2 Function Documentation

7.112.2.1 HxValueList L_HXIMAGEREP HxIdentMaskMoments (HxImageRep *im*, HxImageRep *mask*, HxPoint *p*, HxSizes *size*, int *label*, int *order*)

Compute the moments of all pixels in "im" identified by "mask".

"mask" is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point "p" with given "size" and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation variant, 1 phase export operation with an extra image** (p. 39)
- The pixel functor : **HxExportExtraIdentMaskMoments** (p. 538)
- Instantiations : **HxInstExportExtraIdentMaskMoments_c** (p. **??**)

```
15 {
16     HxValueList valList;
17     HxBoundingBox bb(size);
18     bb = bb.translate(p);
19     HxTagList tags;
20     HxAddTag(tags, "maskVal", label);
21     HxAddTag(tags, "boundingBox", bb);
22     HxAddTag(tags, "order", order);
23     HxAddTag(tags, "valList", &valList);
24     im.exportOpExtra("identMaskMoments", mask, tags);
25     return valList;
26 }
```

7.113 HxIdentMaskStDev.h File Reference

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

Functions

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

  Compute the standard deviation of all pixels in "im" identified by "mask".

7.113.1 Detailed Description

7.113.2 Function Documentation

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

Compute the standard deviation of all pixels in ”im” identified by ”mask”.

”mask” is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point ”p” with given ”size” and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- The pixel functor : **HxInstExportExtraIdentMaskStdev** (p. 932)
- Instantiations : **HxInstExportExtraIdentMaskStdev_c** (p. **??**)

```
14 {
15     HxBoundingBox bb(size);
16     bb = bb.translate(p);
17     HxTagList tags;
18     HxAddTag(tags, "maskVal", label);
19     HxAddTag(tags, "boundingBox", bb);
20     im.exportOpExtra("identMaskStdev", mask, tags);
21     HxValue v = HxGetTag(tags, "result", HxValue(0));
22     return v;
23 }
```

7.114 HxIdentMaskSum.h File Reference

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

Functions

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

  Compute the sum of all pixels in ”im” identified by ”mask”.

7.114.1 Detailed Description

7.114.2 Function Documentation

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

Compute the sum of all pixels in ”im” identified by ”mask”.

”mask” is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point ”p” with given ”size” and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- The pixel functor : **HxInstExportExtraIdentMaskSum** (p. 933)
- Instantiations : **HxInstExportExtraIdentMaskSum_c** (p. **??**)

```
15 {
16     HxBoundingBox bb(size);
17     bb = bb.translate(p);
18     HxTagList tags;
19     HxAddTag(tags, "maskVal", label);
20     HxAddTag(tags, "boundingBox", bb);
21     im.exportOpExtra("identMaskSum", mask, tags);
22     HxValue v = HxGetTag(tags, "result", HxValue(0));
23     return v;
24 }
```

7.115 HxIdentMaskVariance.h File Reference

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

Functions

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

  Compute the variance of all pixels in "im" identified by "mask".

7.115.1 Detailed Description

7.115.2 Function Documentation

7.115.2.1 HxValue L_HXIMAGEREP HxIdentMaskVariance (HxImageRep *im*, HxImageRep *mask*, HxPoint *p*, HxSizes *size*, int *label*)

Compute the variance of all pixels in "im" identified by "mask".

"mask" is assumed to be an identification image with pixel type short. The function considers all pixels within the area starting at point "p" with given "size" and a value equal to label.

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- The pixel functor : **HxInstExportExtraIdentMaskStdev** (p. 932)
- Instantiations : **HxInstExportExtraIdentMaskStdev_c** (p. **??**)

```
15 {
16     HxBoundingBox bb(size);
17     bb = bb.translate(p);
18     HxTagList tags;
19     HxAddTag(tags, "maskVal", label);
20     HxAddTag(tags, "boundingBox", bb);
21     HxAddTag(tags, "exportVariance",true);
```

```
22     im.exportOpExtra("identMaskStdev", mask, tags);
23     HxValue v = HxGetTag(tags, "result", HxValue(0));
24     return v;
25 }
```

7.116 HxImageAsByte.h File Reference

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

Functions

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

  Convert the pixel representation to HxByte.

7.116.1 Detailed Description

7.116.2 Function Documentation

7.116.2.1 HxImageRep L_HXIMAGEREP HxImageAsByte (HxImageRep *img*)

Convert the pixel representation to HxByte.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsByte");
16 
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (img.pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar image", HxGlobalError::HX_GE_I
25         return HxImageRep();
26     }
27 
28     HxImageSignature signature(HXIMAGESIG2DBYTE);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.117 HxImageAsComplex.h File Reference

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

Functions

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

Convert the pixel representation to **HxComplex** (p. 506).

7.117.1 Detailed Description

7.117.2 Function Documentation

7.117.2.1 HxImageRep L_HXIMAGEREP HxImageAsComplex (HxImageRep *img*)

Convert the pixel representation to **HxComplex** (p. 506).

Conversion is done via a cast.

```
15 {
16     HxString fname("HxImageAsComplex");
17
18     if (img.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 2))
24     {
25         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec2 image", HxGlobalError
26         return HxImageRep();
27     }
28
29     HxImageSignature signature(HXIMAGESIG2DCOMPLEX);
30     signature.setImageDimensionality(img.dimensionality());
31     return img.signature() == signature ?
32             img : HxImageFactory::instance().fromImage(signature, img);
33 }
```

7.118 HxImageAsDouble.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsDouble** (**HxImageRep** img)
  Convert the pixel representation to double.

7.118.1 Detailed Description

7.118.2 Function Documentation

7.118.2.1 HxImageRep L_HXIMAGEREP HxImageAsDouble (HxImageRep *img*)

Convert the pixel representation to double.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsDouble");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (img.pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar image", HxGlobalError::HX_GE_I
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DDOUBLE);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.119 HxImageAsFloat.h File Reference

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

Functions

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

  Convert the pixel representation to float.

7.119.1 Detailed Description

7.119.2 Function Documentation

7.119.2.1 HxImageRep L_HXIMAGEREP HxImageAsFloat (HxImageRep *img*)

Convert the pixel representation to float.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsFloat");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (img.pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar image", HxGlobalError::HX_GE_I
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DFLOAT);
29     signature.setImageDimensionality(img.dimensionality());
```

```
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.120 HxImageAsInt.h File Reference

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

Functions

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

  Convert the pixel representation to int.

7.120.1 Detailed Description

7.120.2 Function Documentation

7.120.2.1 HxImageRep L_HXIMAGEREP HxImageAsInt (HxImageRep *img*)

Convert the pixel representation to int.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsInt");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (img.pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar image", HxGlobalError::HX_GE_I
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DINT);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.121 HxImageAsShort.h File Reference

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

Functions

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

Convert the pixel representation to short.

7.121.1 Detailed Description

7.121.2 Function Documentation

7.121.2.1 HxImageRep L_HXIMAGEREP HxImageAsShort (HxImageRep *img*)

Convert the pixel representation to short.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsShort");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (img.pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar image", HxGlobalError::HX_GE_I
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DSHORT);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.122 HxImageAsVec2Byte.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Byte** (**HxImageRep** img)
  Convert the pixel representation to HxVec2Byte.

7.122.1 Detailed Description

7.122.2 Function Documentation

7.122.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Byte (HxImageRep *img*)

Convert the pixel representation to HxVec2Byte.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec2Byte");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 2))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec2 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC2BYTE);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.123 HxImageAsVec2Double.h File Reference

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

Functions

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

  Convert the pixel representation to **HxVec2Double** (p. 1262).

7.123.1 Detailed Description

7.123.2 Function Documentation

7.123.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Double (HxImageRep *img*)

Convert the pixel representation to **HxVec2Double** (p. 1262).

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec2Double");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 2))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec2 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC2DOUBLE);
29     signature.setImageDimensionality(img.dimensionality());
```

```
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.124 HxImageAsVec2Float.h File Reference

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

Functions

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

  Convert the pixel representation to HxVec2Float.

7.124.1 Detailed Description

7.124.2 Function Documentation

7.124.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Float (HxImageRep *img*)

Convert the pixel representation to HxVec2Float.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec2Float");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 2))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec2 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC2FLOAT);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.125 HxImageAsVec2Int.h File Reference

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

Functions

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

Convert the pixel representation to **HxVec2Int** (p. 1281).

7.125.1 Detailed Description

7.125.2 Function Documentation

7.125.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Int (HxImageRep *img*)

Convert the pixel representation to **HxVec2Int** (p. 1281).

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec2Int");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 2))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec2 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC2INT);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.126 HxImageAsVec2Short.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec2Short** (**HxImageRep** img)
  Convert the pixel representation to HxVec2Short.

7.126.1 Detailed Description

7.126.2 Function Documentation

7.126.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec2Short (HxImageRep *img*)

Convert the pixel representation to HxVec2Short.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec2Short");
16 
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 2))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec2 image", HxGlobalError
25         return HxImageRep();
26     }
27 
28     HxImageSignature signature(HXIMAGESIG2DVEC2SHORT);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.127 HxImageAsVec3Byte.h File Reference

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

Functions

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

  Convert the pixel representation to HxVec3Byte.

7.127.1 Detailed Description

7.127.2 Function Documentation

7.127.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Byte (HxImageRep *img*)

Convert the pixel representation to HxVec3Byte.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec3Byte");
16 
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec3 image", HxGlobalError
25         return HxImageRep();
26     }
27 
28 
29     HxImageSignature signature(HXIMAGESIG2DVEC3BYTE);
```

```
30     signature.setImageDimensionality(img.dimensionality());
31     return img.signature() == signature ?
32             img : HxImageFactory::instance().fromImage(signature, img);
33 }
```

7.128 HxImageAsVec3Double.h File Reference

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

Functions

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

  Convert the pixel representation to **HxVec3Double** (p. 1301).

7.128.1 Detailed Description

7.128.2 Function Documentation

7.128.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Double (HxImageRep *img*)

Convert the pixel representation to **HxVec3Double** (p. 1301).

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec3Double");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec3 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC3DOUBLE);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.129 HxImageAsVec3Float.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Float** (**HxImageRep** img)

Convert the pixel representation to HxVec3Float.

7.129.1 Detailed Description

7.129.2 Function Documentation

7.129.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Float (HxImageRep *img*)

Convert the pixel representation to HxVec3Float.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec3Float");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec3 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC3FLOAT);
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.130 HxImageAsVec3Int.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Int** (**HxImageRep** img)
  Convert the pixel representation to **HxVec3Int** (p. 1321).

7.130.1 Detailed Description

7.130.2 Function Documentation

7.130.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Int (HxImageRep *img*)

Convert the pixel representation to **HxVec3Int** (p. 1321).

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec3Int");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec3 image", HxGlobalError
25         return HxImageRep();
26     }
27
28
29     HxImageSignature signature(HXIMAGESIG2DVEC3INT);
30     signature.setImageDimensionality(img.dimensionality());
31     return img.signature() == signature ?
32             img : HxImageFactory::instance().fromImage(signature, img);
33 }
```

7.131 HxImageAsVec3Short.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxImageAsVec3Short** (**HxImageRep** img)

  Convert the pixel representation to HxVec3Short.

7.131.1 Detailed Description

7.131.2 Function Documentation

7.131.2.1 HxImageRep L_HXIMAGEREP HxImageAsVec3Short (HxImageRep *img*)

Convert the pixel representation to HxVec3Short.

Conversion is done via a cast.

```
14 {
15     HxString fname("HxImageAsVec3Short");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((img.pixelDimensionality() != 1) && (img.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar and vec3 image", HxGlobalError
25         return HxImageRep();
26     }
27
28     HxImageSignature signature(HXIMAGESIG2DVEC3SHORT);
```

```
29     signature.setImageDimensionality(img.dimensionality());
30     return img.signature() == signature ?
31             img : HxImageFactory::instance().fromImage(signature, img);
32 }
```

7.132 HxImageMaxSize.h File Reference

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

Functions

- int L_HXIMAGEREP **HxImageMaxSize** (**HxImageRep** img)

  Maximum (one dimensional) size.

7.132.1 Detailed Description

7.132.2 Function Documentation

7.132.2.1 int L_HXIMAGEREP HxImageMaxSize (HxImageRep *img*)

Maximum (one dimensional) size.

Returns Maximum(img.dimensionSize(1), ..., img.dimensionSize(n)) where n == img.dimensionality().

```
13 {
14     HxString fname("HxImageMaxSize");
15 
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21 
22     int i, d = img.dimensionality(), s = img.dimensionSize(1);
23 
24 
25     for (i=2; i<=d; i++)
26         if (img.dimensionSize(i) > s)
27             s = img.dimensionSize(i);
28 
29     return s;
30 }
```

7.133 HxImageMinSize.h File Reference

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

Functions

- int L_HXIMAGEREP **HxImageMinSize** (**HxImageRep** img)

Minimum (one dimensional) size.

7.133.1 Detailed Description

7.133.2 Function Documentation

7.133.2.1 int L_HXIMAGEREP HxImageMinSize (HxImageRep *img*)

Minimum (one dimensional) size.

Returns minimum(img.dimensionSize(1), ..., img.dimensionSize(n)) where n == img.dimensionality().

```
13 {
14     HxString fname("HxImageMaxSize");
15 
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return 0;
20     }
21 
22     int i, d = img.dimensionality(), s = img.dimensionSize(1);
23 
24     for (i=2; i<=d; i++)
25         if (img.dimensionSize(i) < s)
26             s = img.dimensionSize(i);
27 
28     return s;
29 }
```

7.134 HxImagesFromFile.h File Reference

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

Functions

- **HxImageList** L_HXIMAGEREP **HxImagesFromFile** (**HxString** fileName)
  Write images to file using HxImgFileIo.

7.134.1 Detailed Description

7.134.2 Function Documentation

7.134.2.1 HxImageList L_HXIMAGEREP HxImagesFromFile (HxString *fileName*)

Write images to file using HxImgFileIo.

```
15 {
16     HxTagList tags;
17     return HxImageFactory::instance().imagesFromFile(fileName, tags);
18 }
```

7.135 HxImagesToFile.h File Reference

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

Functions

- bool L_HXIMAGEREP **HxImagesToFile** (**HxImageList** ims, **HxString** fileName)

  Write images to file using HxImgFileIo.

7.135.1 Detailed Description

7.135.2 Function Documentation

7.135.2.1 bool L_HXIMAGEREP HxImagesToFile (HxImageList *ims*, HxString *fileName*)

Write images to file using HxImgFileIo.

```
15 {
16 /*
17 Errors
18 */
19     HxTagList tags;
20     return HxImageFactory::instance().imagesToFile(ims, fileName, tags);
21 }
```

7.136 HxImageToSegmentation.h File Reference

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

Functions

- **HxSegmentation2d** ∗ **HxImageToSegmentation** (**HxImageRep** input, **HxImageRep** segments, int connectivity, int minArea)

  Convert image segments to an **HxSegmentation2d** (p. 1182).

7.136.1 Detailed Description

7.136.2 Function Documentation

7.136.2.1 HxSegmentation2d∗ HxImageToSegmentation (HxImageRep *input*, HxImageRep *segments*, int *connectivity*, int *minArea*)

Convert image segments to an **HxSegmentation2d** (p. 1182).

```
259 {
260     HxTagList tags;
261     HxAddTag(tags, "connectivity", connectivity);
```

```
262     HxAddTag(tags, "minArea", minArea);
263
264     HxSegmentation2d* seg = new HxSegmentation2d();
265     seg->setInputImage(input);
266     HxAddTag(tags, "segmentation", seg);
267
268     HxImageRep labeled = segments.queueBasedOp(segments, "ImageToSegmentation",
269                                                tags);
270     seg->setLabeledImage(labeled);
271
272     // because in filling the neigbours list, we were looking only in the past,
273     // not in the future we still have to make the list of neigbours symetric
274     HxBlob2dRelation* relation = seg->getRelation("neighbours");
275     for (HxBlob2dListConstIter bi=relation->getBlobBegin() ;
276                                bi<relation->getBlobEnd() ; bi++) {
277         HxBlob2dListConstIter bj = bi;
278         bj++;
279         HxBlob2dListBackInserter ins = relation->findRelatedBlobsInserter(*bi);
280         for ( ; bj<relation->getBlobEnd() ; bj++) {
281             for (HxBlob2dListConstIter bk = relation->findRelatedBlobsBegin(*bj) ;
282                                        bk < relation->findRelatedBlobsEnd(*bj) ;
283                                        bk++) {
284                 if ((*bi)->getLabel() == (*bk)->getLabel()) {
285                     *ins++ = *bj;
286                 }
287             }
288         }
289     }
290     return seg;
291 }
```

7.137 HxInf.h File Reference

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

Functions

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

  Inifimum.

7.137.1 Detailed Description

7.137.2 Function Documentation

7.137.2.1 HxImageRep L_HXIMAGEREP HxInf (HxImageRep *im1*, HxImageRep *im2*)

Inifimum.

The function performs infimum (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoInf** (p. 436). The image functor instantiator : **HxInstantiatorInf** (p. 886).

```
14 {
```

```
15     HxString fname("HxInf");
16
17     if (im1.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (im2.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27
28     if (im1.dimensionality() != im2.dimensionality())
29     {
30         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
31         return HxImageRep();
32     }
33     if (im1.pixelDimensionality() != im2.pixelDimensionality())
34     {
35         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
36         return HxImageRep();
37     }
38
39     if (im1.sizes().x() != im2.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im1.sizes().y() != im2.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (im1.dimensionality() > 2)
50     {
51         if (im1.sizes().z() != im2.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGE
54             return HxImageRep();
55         }
56     }
57
58     return im1.binaryPixOp(im2, "inf");
59 }
```

7.138 HxInfimumReconstruction.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxInfimumReconstruction** (**HxImageRep** im, **HxImageRep** mask, **HxSF** sf)

  function y=mminfrec_equ(f, g, bc) n = length(f); y = mmcdil(f,g,bc,n);.

7.138.1 Detailed Description

7.138.2 Function Documentation

7.138.2.1 HxImageRep L_HXIMAGEREP HxInfimumReconstruction (HxImageRep *im*, HxImageRep *mask*, HxSF *sf*)

function y=mminfrec_equ(f, g, bc) n = length(f); y = mmcdil(f,g,bc,n);.

```
25 {
26     HxImageRep res;
27
28     //this might be a huge number.
29     //defintion is to do conditional dilation untill stability
30     //I should check for stability here, or in ConditionalDilation
31     //by comparing two subsequent images
32     //Because this is also expesinve, I might check only after every 10 iterations
33
34     int n = im.dimensionSize(1)*im.dimensionSize(2);
35
36     res = HxConditionalDilation(im, mask, sf, n);
37     return res;
38 }
```

7.139 HxInfVal.h File Reference

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

Functions

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

  Inifimum.

7.139.1 Detailed Description

7.139.2 Function Documentation

7.139.2.1 HxImageRep L_HXIMAGEREP HxInfVal (HxImageRep *im*, HxValue *val*)

Inifimum.

The function performs infimum (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoInf** (p. 436). The image functor instantiator : **HxInstantiatorInfV** (p. 887).

```
13 {
14     HxString fname("HxInfVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
```

```
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if (im.signature().pixelDimensionality() != valdim)
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "inf");
43 }
```

7.140 HxInverseProjectRange.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxInverseProjectRange** (**HxImageRep** im, int dimension, **HxImageRep** arg)

  Inverse projection of the pixel range.

7.140.1 Detailed Description

7.140.2 Function Documentation

7.140.2.1 HxImageRep L_HXIMAGEREP HxInverseProjectRange (HxImageRep *im*, int *dimension*, HxImageRep *arg*)

Inverse projection of the pixel range.

The function projects (see **Pixels** (p. 3)) all pixels of image im on the given dimension of image arg via a unary pixel operation (see **Images** (p. 8)). Dimension starts at 1.

```
13 {
14     HxString fname("HxInverseProjectRange");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
```

```
20      }
21      if (arg.isNull())
22      {
23          HxGlobalError::instance()->reportError(fname, arg.name(), "null arg image", HxGlobalError::HX_GE_INVALID);
24          return HxImageRep();
25      }
26
27      if (im.dimensionality() != arg.dimensionality())
28      {
29          HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30          return HxImageRep();
31      }
32
33      if (im.pixelDimensionality() != 1)
34      {
35          HxGlobalError::instance()->reportError(fname, "operation only valid on scalar input images", HxGlobalError::H
36          return HxImageRep();
37      }
38
39      if (im.sizes().x() != arg.sizes().x())
40      {
41          HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42          return HxImageRep();
43      }
44      if (im.sizes().y() != arg.sizes().y())
45      {
46          HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47          return HxImageRep();
48      }
49      if (im.dimensionality() > 2)
50      {
51          if (im.sizes().z() != arg.sizes().z())
52          {
53              HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
54              return HxImageRep();
55          }
56      }
57      if (dimension < 1)
58      {
59          HxGlobalError::instance()->reportError(fname, "dimension parameter should be greater than zero", HxGlobalErro
60          return HxImageRep();
61      }
62      if (dimension > arg.pixelDimensionality())
63      {
64          HxGlobalError::instance()->reportError(fname, "dimension parameter should be less than result pixel dimensio
65          return HxImageRep();
66      }
67
68      return im.inverseProjectRange(dimension, arg);
69  }
```

7.141 HxKuwahara.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxKuwahara** (**HxImageRep** im, int width, int height)

  Performs the Kuwahara Filter.

7.141.1 Detailed Description

7.141.2 Function Documentation

7.141.2.1 HxImageRep L_HXIMAGEREP HxKuwahara (HxImageRep *im*, int *width*, int *height*)

Performs the Kuwahara Filter.

This filter is an edge-preserving filter.

(a a ab b b) (a a ab b b) (ac ac abcd bd bd) (c c cd d d) (c c cd d d)

In each of the four regions (a, b, c, d), the mean brightness and the variance are calculated. The output value of the center pixel (abcd) in the window is the mean value of that region that has the smallest variance. This filter is an edge-preserving filter, which smoothes the images without disturbing the sharpness and the position of edges.

```
12 {
13     //the window width and height have to be ODD
14     HxTagList tags;
15     HxAddTag(tags, "windowW", width);
16     HxAddTag(tags, "windowH", height);
17
18     return im.neighbourhoodOp("kuwahara", tags);
19 }
```

7.142 HxLabel.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxLabel** (**HxImageRep** input, int conn)

  Label function based on GrassFire.

7.142.1 Detailed Description

7.142.2 Function Documentation

7.142.2.1 HxImageRep L_HXIMAGEREP HxLabel (HxImageRep *input*, int *conn*)

Label function based on GrassFire.

```
179 {
180     HxTagList tags;
181     HxAddTag(tags, "connectivity", conn);
182
183     return input.queueBasedOp(input, "qLabelGrassFire", tags);
184 }
```

7.143 HxLabel2.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxLabel2** (**HxImageRep** input, int conn)
  Label function (alternative).

7.143.1 Detailed Description

7.143.2 Function Documentation

7.143.2.1 HxImageRep L_HXIMAGEREP HxLabel2 (HxImageRep *input*, int *conn*)

Label function (alternative).

```
207 {
208     HxTagList tags;
209     HxAddTag(tags, "connectivity", conn);
210
211     return input.queueBasedOp(input, "qLabelFrans", tags);
212 }
```

7.144 HxLeftShift.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxLeftShift** (**HxImageRep** im1, **HxImageRep** im2)
  Left shift.

7.144.1 Detailed Description

7.144.2 Function Documentation

7.144.2.1 HxImageRep L_HXIMAGEREP HxLeftShift (HxImageRep *im1*, HxImageRep *im2*)

Left shift.

The function performs left shift (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoLeftShift** (p. 439). The image functor instantiator : **HxInstantiatorLeftShift** (p. 888).

```
13 {
14     HxString fname("HxLeftShift");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if (im1.pixelDimensionality() != im2.pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35         return HxImageRep();
36     }
37     if (im1.pixelDimensionality() != 1)
38     {
39         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
40         return HxImageRep();
41     }
42 
43     if (im1.signature().pixelType() != im2.signature().pixelType())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.signature().pixelType() != INT_VALUE)
49     {
50         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer values", HxGlobalE
51         return HxImageRep();
52     }
53 
54     if (im1.sizes().x() != im2.sizes().x())
55     {
56         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
57         return HxImageRep();
58     }
59     if (im1.sizes().y() != im2.sizes().y())
60     {
61         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
62         return HxImageRep();
63     }
64     if (im1.signature().imageDimensionality() > 2)
65     {
66         if (im1.sizes().z() != im2.sizes().z())
67         {
68             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
69             return HxImageRep();
70         }
71     }
72     return im1.binaryPixOp(im2, "leftShift");
73 }
```

7.145 HxLeftShiftVal.h File Reference

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

Functions

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

  Left shift.

7.145.1 Detailed Description

7.145.2 Function Documentation

7.145.2.1 HxImageRep L_HXIMAGEREP HxLeftShiftVal (HxImageRep *im*, HxValue *val*)

Left shift.

The function performs left shift (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoLeftShift** (p. 439). The image functor instantiator : **HxInstantiatorLeftShiftV** (p. 889).

```
13 {
14     HxString fname("HxLeftShiftVal");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21 
22     if (im.signature().pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
25         return HxImageRep();
26     }
27 
28     if (im.signature().pixelType() != INT_VALUE)
29     {
30         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
31         return HxImageRep();
32     }
33 
34     if (val.tag() != HxValue::SI)
35     {
36         HxGlobalError::instance()->reportError(fname, "only scalar integer value supported", HxGlobalError::HX_GE_INV
37         return HxImageRep();
38     }
39 
40 
41     return im.binaryPixOp(val, "leftShift");
42 }
```

7.146 HxLessEqual.h File Reference

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

Functions

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

  Less equal.

7.146.1 Detailed Description

7.146.2 Function Documentation

7.146.2.1 HxImageRep L_HXIMAGEREP HxLessEqual (HxImageRep *im1*, HxImageRep *im2*)

Less equal.

The function performs less equal (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoLessEqual** (p. 440). The image functor instantiator : **HxInstantiatorLessEqual** (p. 889).

```
13 {
14     HxString fname("HxLessEqual");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32     if ((im1.pixelDimensionality() != 1) || (im2.pixelDimensionality() != 1))
33     {
34         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
35         return HxImageRep();
36     }
37 
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
```

```
47     }
48     if (im1.dimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
54         }
55     }
56
57     return im1.binaryPixOp(im2, "lessEqual");
58 }
```

7.147 HxLessEqualVal.h File Reference

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

Functions

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

  Less equal.

7.147.1 Detailed Description

7.147.2 Function Documentation

7.147.2.1 HxImageRep L_HXIMAGEREP HxLessEqualVal (HxImageRep *im*, HxValue *val*)

Less equal.

The function performs less equal (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoLessEqual** (p. 440). The image functor instantiator : **HxInstantiatorLessEqualV** (p. 890).

```
13 {
14     HxString fname("HxLessEqualVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
24         return HxImageRep();
25     }
26     if ((val.tag() != HxValue::SI) && (val.tag() != HxValue::SD))
27     {
28         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
29         return HxImageRep();
30     }
31
```

```
32     return im.binaryPixOp(val, "lessEqual");
33 }
```

7.148 HxLessThan.h File Reference

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

Functions

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

  Less than.

7.148.1 Detailed Description

7.148.2 Function Documentation

7.148.2.1 HxImageRep L_HXIMAGEREP HxLessThan (HxImageRep *im1*, HxImageRep *im2*)

Less than.

The function performs less than (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoLessThan** (p. 441). The image functor instantiator : **HxInstantiatorLessThan** (p. 890).

```
13 {
14     HxString fname("HxLessThan");
15
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if ((im1.pixelDimensionality() != 1) || (im2.pixelDimensionality() != 1))
33     {
34         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
35         return HxImageRep();
36     }
37
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
```

```
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.dimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
54         }
55     }
56
57     return im1.binaryPixOp(im2, "lessThan");
58 }
```

7.149 HxLessThanVal.h File Reference

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

Functions

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

  Less than.

7.149.1 Detailed Description

7.149.2 Function Documentation

7.149.2.1 HxImageRep L_HXIMAGEREP HxLessThanVal (HxImageRep *im*, HxValue *val*)

Less than.

The function performs less than (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoLessThan** (p. 441). The image functor instantiator : **HxInstantiatorLessThanV** (p. 891).

```
13 {
14     HxString fname("HxLessThanVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
24         return HxImageRep();
25     }
26     if ((val.tag() != HxValue::SI) && (val.tag() != HxValue::SD))
```

```
27     {
28         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_UI
29         return HxImageRep();
30     }
31
32     return im.binaryPixOp(val, "lessThan");
33 }
```

7.150 HxLocalMode.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxLocalMode** (**HxImageRep** f, **HxImageRep** g, int nr, double sigmax, double sigmay, double sigmaval, **HxSizes** ngbSize)

  Local mode filter.

7.150.1 Detailed Description

7.150.2 Function Documentation

7.150.2.1 HxImageRep L_HXIMAGEREP HxLocalMode (HxImageRep *f*, HxImageRep *g*, int *nr*, double *sigmax*, double *sigmay*, double *sigmaval*, HxSizes *ngbSize*)

Local mode filter.

```
15 {
16     HxTagList tags;
17     HxAddTag(tags, "sigmax", sigmax);
18     HxAddTag(tags, "sigmay", sigmay);
19     HxAddTag(tags, "sigmaval", sigmaval);
20     HxAddTag(tags, "ngbSize", ngbSize);
21     for (int i=0; i<nr; i++)
22         g=f.neighbourhoodOpExtra("localMode", g, tags);
23     return g;
24 }
```

7.151 HxLog.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxLog** (**HxImageRep** im)

  Natural logarithm.

7.151.1 Detailed Description

7.151.2 Function Documentation

7.151.2.1 HxImageRep L_HXIMAGEREP HxLog (HxImageRep *im*)

Natural logarithm.

The function computes the natural logarithm (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoLog** (p. 1226). The image functor instantiator : **HxInstantiatorLog** (p. 892).

```
13 {
14     HxString fname("HxLog");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     HxValue vinf = HxPixInf(im);
23
24     if (im.signature().pixelDimensionality() == 1)
25     {
26         if (((HxScalarDouble) vinf) <= 0.0)
27         {
28             HxGlobalError::instance()->reportError(fname, im.name(), "values less than 0", HxGlobalError::HX_GE_OUTOF
29         }
30     }
31     else if (im.signature().pixelDimensionality() == 2)
32     {
33         HxVec2Double vinf2d = (HxVec2Double) vinf;
34         if ((vinf2d.x() <= 0.0) || (vinf2d.y() <= 0.0))
35         {
36             HxGlobalError::instance()->reportError(fname, im.name(), "2D values are less than 0", HxGlobalError::HX_G
37         }
38     }
39     else if (im.signature().pixelDimensionality() == 3)
40     {
41         HxVec3Double vinf3d = (HxVec3Double) vinf;
42         if ((vinf3d.x() <= 0.0) || (vinf3d.y() <= 0.0) || (vinf3d.z() <= 0.0))
43         {
44             HxGlobalError::instance()->reportError(fname, im.name(), "3D values are less than 0", HxGlobalError::HX_G
45         }
46     }
47
48     return im.unaryPixOp("log");
49 }
```

7.152 HxLog10.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxLog10** (**HxImageRep** im)

Base 10 logarithm.

7.152.1 Detailed Description

7.152.2 Function Documentation

7.152.2.1 HxImageRep L_HXIMAGEREP HxLog10 (HxImageRep *im*)

Base 10 logarithm.

The function computes the base 10 logarithm (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoLog10** (p. 1227). The image functor instantiator : **HxInstantiatorLog10** (p. 892).

```
13 {
14     HxString fname("HxLog10");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     HxValue vinf = HxPixInf(im);
22
23     if (im.signature().pixelDimensionality() == 1)
24     {
25         if (((HxScalarDouble) vinf) <= 0.0)
26         {
27             HxGlobalError::instance()->reportError(fname, im.name(), "values less than 0", HxGlobalError::HX_GE_OUTO
28         }
29     }
30     else if (im.signature().pixelDimensionality() == 2)
31     {
32         HxVec2Double vinf2d = (HxVec2Double) vinf;
33         if ((vinf2d.x() <= 0.0) || (vinf2d.y() <= 0.0))
34         {
35             HxGlobalError::instance()->reportError(fname, im.name(), "2D values are less than 0", HxGlobalError::HX_
36         }
37     }
38     else if (im.signature().pixelDimensionality() == 3)
39     {
40         HxVec3Double vinf3d = (HxVec3Double) vinf;
41         if ((vinf3d.x() <= 0.0) || (vinf3d.y() <= 0.0) || (vinf3d.z() <= 0.0))
42         {
43             HxGlobalError::instance()->reportError(fname, im.name(), "3D values are less than 0", HxGlobalError::HX_
44         }
45     }
46
47     return im.unaryPixOp("log10");
48 }
```

7.153 HxLUT.h File Reference

```
#include "HxImageRep.h"
#include <map>
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxLUT** (**HxImageRep** im, std::map< int, int > *lut)

  this function implements the Look Up Table (LUT) operation the user has to prepare the map of equivalences.

7.153.1 Detailed Description

7.153.2 Function Documentation

7.153.2.1 HxImageRep L_HXIMAGEREP HxLUT (HxImageRep *im*, std::map< int, int > * *lut*)

this function implements the Look Up Table (LUT) operation the user has to prepare the map of equivalences.

<oldValue, newValue> pass the pointer to this map<int,int> Remark: for 3DByte pixel type, we encode the 3 values into an int as: (z<<16) | (y<<8) | x

```
143 {
144     HxImageRep out;
145 
146     HxTagList tags;
147     HxAddTag(tags,"LUTPointer",lut);
148     out = im.unaryPixOp("lut", tags);
149 
150     return out;
151 }
```

7.154 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.

7.154.1 Detailed Description

7.154.2 Function Documentation

7.154.2.1 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     HxString fname("HxMakeFrom2Images");
16 
17     if (i1.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, i1.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (i2.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, i2.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27 
28     if (i1.dimensionality() != i2.dimensionality())
29     {
30         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
31         return HxImageRep();
32     }
33     if (i1.pixelType() != i2.pixelType())
34     {
35         HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
36         return HxImageRep();
37     }
38     if (i1.pixelDimensionality() != i2.pixelDimensionality())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
41         return HxImageRep();
42     }
43 
44     if (i1.sizes().x() != i2.sizes().x())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (i1.sizes().y() != i2.sizes().y())
50     {
51         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
52         return HxImageRep();
53     }
54     if (i1.dimensionality() > 2)
55     {
56         if (i1.sizes().z() != i2.sizes().z())
57         {
58             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
59             return HxImageRep();
60         }
61     }
62 
63     return HxImageFactory::instance().from2Images(i1, i2);
64 }
```

7.155 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.

7.155.1 Detailed Description

7.155.2 Function Documentation

7.155.2.1 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.

```
14 {
15     HxString fname("HxMakeFrom3Images");
16 
17     if (i1.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, i1.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (i2.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, i2.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27     if (i3.isNull())
28     {
29         HxGlobalError::instance()->reportError(fname, i3.name(), "null image", HxGlobalError::HX_GE_INVALID);
30         return HxImageRep();
31     }
32 
33     if ((i1.dimensionality() != i2.dimensionality()) || (i1.dimensionality() != i3.dimensionality()))
34     {
35         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
36         return HxImageRep();
37     }
38     if ((i1.pixelType() != i2.pixelType()) || (i1.pixelType() != i3.pixelType()))
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if ((i1.pixelDimensionality() != i2.pixelDimensionality()) || (i1.pixelDimensionality() != i3.pixelDimensionality
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
46         return HxImageRep();
47     }
48 
49     if ((i1.sizes().x() != i2.sizes().x()) || (i1.sizes().x() != i3.sizes().x()))
50     {
51         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
52         return HxImageRep();
53     }
54     if ((i1.sizes().y() != i2.sizes().y()) || (i1.sizes().y() != i3.sizes().y()))
55     {
56         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
57         return HxImageRep();
58     }
59     if (i1.dimensionality() > 2)
```

```
60     {
61         if ((i1.sizes().z() != i2.sizes().z()) || (i1.sizes().z() != i3.sizes().z()))
62         {
63             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
64             return HxImageRep();
65         }
66     }
67
68     return HxImageFactory::instance().from3Images(i1, i2, i3);
69 }
```

7.156 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.

7.156.1 Detailed Description

7.156.2 Function Documentation

7.156.2.1 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     HxString fname("HxMakeFromByteData");
17
18     if ((pixelDimensionality < 1) || (pixelDimensionality > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((dimensions < 2) || (dimensions > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
```

```
38     }
39     if (dimensions > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47     // check size of HxByte*, but how?
48
49     return HxImageFactory::instance().fromByteData(pixelDimensionality,
50                 dimensions, sizes, data);
51 }
```

7.157 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.

7.157.1 Detailed Description

7.157.2 Function Documentation

7.157.2.1 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     HxString fname("HxMakeFromDoubleData");
17
18     if ((pixelDimensionality < 1) || (pixelDimensionality > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((dimensions < 2) || (dimensions > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
```

```
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
38     }
39     if (dimensions > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47     // check size of double*, but how?
48
49     return HxImageFactory::instance().fromDoubleData(pixelDimensionality,
50                 dimensions, sizes, data);
51 }
```

7.158 HxMakeFromFile.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromFile** (**HxString** fileName)

  New image read from file using HxImgFileIo lib.

7.158.1 Detailed Description

7.158.2 Function Documentation

7.158.2.1 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 }
```

7.159 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.

7.159.1 Detailed Description

7.159.2 Function Documentation

7.159.2.1 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     HxString fname("HxMakeFromFloatData");
17
18     if ((pixelDimensionality < 1) || (pixelDimensionality > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((dimensions < 2) || (dimensions > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
38     }
39     if (dimensions > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47     // check size of float*, but how?
48
49     return HxImageFactory::instance().fromFloatData(pixelDimensionality,
50                 dimensions, sizes, data);
51 }
```

7.160 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.

7.160.1 Detailed Description

7.160.2 Function Documentation

7.160.2.1 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 }
```

7.161 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.

7.161.1 Detailed Description

7.161.2 Function Documentation

7.161.2.1 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     HxString fname("HxMakeFromGrayValue");
17 
18     if ((signature.pixelDimensionality() < 1) || (signature.pixelDimensionality() > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((signature.imageDimensionality() < 2) || (signature.imageDimensionality() > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
```

```
28
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
38     }
39     if (signature.imageDimensionality() > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47 // check size of HxByte*, but how?
48
49     return HxImageFactory::instance().fromGrayValue(signature, sizes, pixels);
50 }
```

7.162 HxMakeFromImage.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromImage** (const **HxImageSignature** &signature, **HxImageRep** src)

  Make a new image with given signature.

7.162.1 Detailed Description

7.162.2 Function Documentation

7.162.2.1 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     HxString fname("HxMakeFromImage");
16
17     if (src.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, src.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22
23     return HxImageFactory::instance().fromImage(signature, src);
24 }
```

7.163 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.

7.163.1 Detailed Description

7.163.2 Function Documentation

7.163.2.1 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 }
```

7.164 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.

7.164.1 Detailed Description

7.164.2 Function Documentation

7.164.2.1 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     HxString fname("HxMakeFromIntData");
17
18     if ((pixelDimensionality < 1) || (pixelDimensionality > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((dimensions < 2) || (dimensions > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
38     }
39     if (dimensions > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47     // check size of int*, but how?
48
49     return HxImageFactory::instance().fromIntData(
50             pixelDimensionality, dimensions, sizes, data);
51 }
```

7.165 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.

7.165.1 Detailed Description

7.165.2 Function Documentation

7.165.2.1 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     HxString fname("HxMakeFromJavaRgb");
17
18     if ((signature.pixelDimensionality() < 1) || (signature.pixelDimensionality() > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((signature.imageDimensionality() < 2) || (signature.imageDimensionality() > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
38     }
39     if (signature.imageDimensionality() > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47 // check size of int*, but how?
48
49
50     return HxImageFactory::instance().fromJavaRgb(signature, sizes, pixels);
51 }
```

7.166 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.

7.166.1 Detailed Description

7.166.2 Function Documentation

7.166.2.1 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     HxString fname("HxMakeFromMatlab");
17 
18     if ((signature.pixelDimensionality() < 1) || (signature.pixelDimensionality() > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((signature.imageDimensionality() < 2) || (signature.imageDimensionality() > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28 
29     if (sizes.x() < 1)
30     {
31         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
32         return HxImageRep();
33     }
34     if (sizes.y() < 1)
35     {
36         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
37         return HxImageRep();
38     }
39     if (signature.imageDimensionality() > 2)
40     {
41         if (sizes.y() < 1)
42         {
43             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
44             return HxImageRep();
45         }
46     }
47 // check size of double*, but how?
48 
49     return HxImageFactory::instance().fromMatlab(signature, sizes, pixels);
50 }
```

7.167 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.

7.167.1 Detailed Description

7.167.2 Function Documentation

7.167.2.1 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     HxString fname("HxMakeFromNamedGenerator");
17
18     if ((signature.pixelDimensionality() < 1) || (signature.pixelDimensionality() > 3))
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if ((signature.imageDimensionality() < 2) || (signature.imageDimensionality() > 3))
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28 // check generator
29
30     return HxImageFactory::instance().fromNamedGenerator(signature,
31                 generatorName, tags);
32 }
```

7.168 HxMakeFromPpmPixels.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxMakeFromPpmPixels** (const **HxImageSignature** &signature, **HxSizes** sizes, const **HxByte** *pixels)

7.168.1 Detailed Description

7.169 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.

7.169.1 Detailed Description

7.169.2 Function Documentation

7.169.2.1 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 }
```

7.170 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.

7.170.1 Detailed Description

7.170.2 Function Documentation

7.170.2.1 HxImageRep L_HXIMAGEREP HxMakeFromSignature (const HxImageSignature & *signature*, HxSizes *sizes*)

Make an uninitialized image with given signature and sizes.

```
14 {
15     HxString fname("HxMakeFromSignature");
16 
17     if ((signature.pixelDimensionality() < 1) || (signature.pixelDimensionality() > 3))
18     {
19         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((signature.imageDimensionality() < 2) || (signature.imageDimensionality() > 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27 
28     if (sizes.x() < 1)
29     {
30         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
31         return HxImageRep();
```

```
32     }
33     if (sizes.y() < 1)
34     {
35         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
36         return HxImageRep();
37     }
38     if (signature.imageDimensionality() > 2)
39     {
40         if (sizes.y() < 1)
41         {
42             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
43             return HxImageRep();
44         }
45     }
46 // check size of int*, but how?
47
48     return HxImageFactory::instance().fromSignature(signature, sizes);
49 }
```

7.171 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.

7.171.1 Detailed Description

7.171.2 Function Documentation

7.171.2.1 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     HxString fname("HxMakeFromValue");
16
17     if ((signature.pixelDimensionality() < 1) || (signature.pixelDimensionality() > 3))
18     {
19         HxGlobalError::instance()->reportError(fname, "Illegal pixel dimensionality", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if ((signature.imageDimensionality() < 2) || (signature.imageDimensionality() > 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "Illegal number of dimensions", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27
28     if (sizes.x() < 1)
```

```
29     {
30         HxGlobalError::instance()->reportError(fname, "Illegal x size", HxGlobalError::HX_GE_INVALID);
31         return HxImageRep();
32     }
33     if (sizes.y() < 1)
34     {
35         HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
36         return HxImageRep();
37     }
38     if (signature.imageDimensionality() > 2)
39     {
40         if (sizes.y() < 1)
41         {
42             HxGlobalError::instance()->reportError(fname, "Illegal y size", HxGlobalError::HX_GE_INVALID);
43             return HxImageRep();
44         }
45     }
46
47     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
48     {
49         if (signature.pixelDimensionality() != 1)
50         {
51             HxGlobalError::instance()->reportError(fname, "value dimensionality does not correspond to signature", Hx
52             return HxImageRep();
53         }
54     }
55     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D)
56             || (val.tag() == HxValue::CPL))
57     {
58         if (signature.pixelDimensionality() != 2)
59         {
60             HxGlobalError::instance()->reportError(fname, "value dimensionality does not correspond to signature", Hx
61             return HxImageRep();
62         }
63     }
64     else if ((val.tag() == HxValue::V3I) || (val.tag() == HxValue::V3D))
65     {
66         if (signature.pixelDimensionality() != 3)
67         {
68             HxGlobalError::instance()->reportError(fname, "value dimensionality does not correspond to signature", Hx
69             return HxImageRep();
70         }
71     }
72
73     return HxImageFactory::instance().fromValue(signature, sizes, val);
74 }
```

7.172 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.

7.172.1 Detailed Description

7.172.2 Function Documentation

7.172.2.1 HxImageRep L_HXIMAGEREP HxMakeGaussian1d (double *sigma*, int *deri*, double *acc*, int *maxfsize*, int *fsize* = -1)

Generate a kernel image resembling a Gaussian.

```
15 {
16     HxString fname("HxMakeGaussian1d");
17
18     if (sigma <= 0)
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal value of sigma", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if (deri < 0)
24     {
25         HxGlobalError::instance()->reportError(fname, "Illegal value of deri", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28     if (acc < 0)
29     {
30         HxGlobalError::instance()->reportError(fname, "Illegal value of acc", HxGlobalError::HX_GE_INVALID);
31         return HxImageRep();
32     }
33     if (maxfsize < 1)
34     {
35         HxGlobalError::instance()->reportError(fname, "Illegal value of maxfsize (>=1)", HxGlobalError::HX_GE_INVALI
36         return HxImageRep();
37     }
38
39     HxTagList   tags;
40
41     HxAddTag(tags, "sigma", sigma);
42     HxAddTag(tags, "derivative", deri);
43     HxAddTag(tags, "accuracy", acc);
44     HxAddTag(tags, "size", fsize);
45     HxAddTag(tags, "maxSize", maxfsize);
46
47     return HxImageFactory::instance().fromNamedGenerator(
48             HXIMAGESIG2DDOUBLE, "gauss1d", tags);
49 }
```

7.173 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.

7.173.1 Detailed Description

7.173.2 Function Documentation

7.173.2.1 HxImageRep L_HXIMAGEREP HxMakeParabola1d (double *rho*, double *accuracy*, int *maxfsize*, int *fsize* = -1)

Generate a kernel image resembling a parabola.

```
15 {
16     HxString fname("HxMakeParabola1d");
17
18     if (acc < 0)
19     {
20         HxGlobalError::instance()->reportError(fname, "Illegal value of accuracy", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23
24     if (maxfsize < 1)
25     {
26         HxGlobalError::instance()->reportError(fname, "Illegal value of maxfsize (>=1)", HxGlobalError::HX_GE_INVALI
27         return HxImageRep();
28     }
29
30     HxTagList   tags;
31
32     HxAddTag(tags, "rho", HxVec3Double(rho,rho,rho));
33     HxAddTag(tags, "accuracy", acc);
34     HxAddTag(tags, "size", fsize);
35     HxAddTag(tags, "maxSize", maxfsize);
36
37     return HxImageFactory::instance().fromNamedGenerator(
38             HXIMAGESIG2DDOUBLE, "parabola1d", tags);
39 }
```

7.174 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)

7.174.1 Detailed Description

7.175 HxMax.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxMax** (**HxImageRep** im1, **HxImageRep** im2)
  Maximum.

7.175.1 Detailed Description

7.175.2 Function Documentation

7.175.2.1 HxImageRep L_HXIMAGEREP HxMax (HxImageRep *im1*, HxImageRep *im2*)

Maximum.

The function performs maximum (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMax** (p. 443). The image functor instantiator : **HxInstantiatorMax** (p. 893).

```
14 {
15     HxString fname("HxMax");
16
17     if (im1.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (im2.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27
28     if (im1.dimensionality() != im2.dimensionality())
29     {
30         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
31         return HxImageRep();
32     }
33     if ((im1.pixelDimensionality() != 1) || (im2.pixelDimensionality() != 1))
34     {
35         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
36         return HxImageRep();
37     }
38
39     if (im1.sizes().x() != im2.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im1.sizes().y() != im2.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (im1.dimensionality() > 2)
50     {
51         if (im1.sizes().z() != im2.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
54             return HxImageRep();
```

```
55         }
56     }
57
58     return im1.binaryPixOp(im2, "max");
59 }
```

7.176 HxMaxVal.h File Reference

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

Functions

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

  Maximum.

7.176.1 Detailed Description

7.176.2 Function Documentation

7.176.2.1 HxImageRep L_HXIMAGEREP HxMaxVal (HxImageRep *im*, HxValue *val*)

Maximum.

The function performs maximum (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMax** (p. 443). The image functor instantiator : **HxInstantiatorMaxV** (p. 894).

```
13 {
14     HxString fname("HxMaxVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_UI
24         return HxImageRep();
25     }
26     if ((val.tag() != HxValue::SI) && (val.tag() != HxValue::SD))
27     {
28         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_UI
29         return HxImageRep();
30     }
31
32     return im.binaryPixOp(val, "max");
33 }
```

7.177 HxMin.h File Reference

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

Functions

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

  Minimum.

7.177.1 Detailed Description

7.177.2 Function Documentation

7.177.2.1 HxImageRep L_HXIMAGEREP HxMin (HxImageRep *im1*, HxImageRep *im2*)

Minimum.

The function performs minimum (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMin** (p. 446). The image functor instantiator : **HxInstantiatorMin** (p. 895).

```
14 {
15     HxString fname("HxMin");
16 
17     if (im1.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (im2.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27 
28     if (im1.dimensionality() != im2.dimensionality())
29     {
30         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
31         return HxImageRep();
32     }
33     if ((im1.pixelDimensionality() != 1) || (im2.pixelDimensionality() != 1))
34     {
35         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
36         return HxImageRep();
37     }
38 
39     if (im1.sizes().x() != im2.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im1.sizes().y() != im2.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
```

```
48     }
49     if (im1.dimensionality() > 2)
50     {
51         if (im1.sizes().z() != im2.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
54             return HxImageRep();
55         }
56     }
57
58      return im1.binaryPixOp(im2, "min");
59 }
```

7.178 HxMinVal.h File Reference

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

Functions

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

  Minimum.

7.178.1 Detailed Description

7.178.2 Function Documentation

7.178.2.1 HxImageRep L_HXIMAGEREP HxMinVal (HxImageRep *im*, HxValue *val*)

Minimum.

The function performs minimum (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMin** (p. 446). The image functor instantiator : **HxInstantiatorMinV** (p. 896).

```
13 {
14     HxString fname("HxMinVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
24         return HxImageRep();
25     }
26     if ((val.tag() != HxValue::SI) && (val.tag() != HxValue::SD))
27     {
28         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar types", HxGlobalError::HX_GE_U
29         return HxImageRep();
30     }
31
```

```
32     return im.binaryPixOp(val, "min");
33 }
```

7.179 HxMod.h File Reference

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

Functions

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

  Modulo.

7.179.1 Detailed Description

7.179.2 Function Documentation

7.179.2.1 HxImageRep L_HXIMAGEREP HxMod (HxImageRep *im1*, HxImageRep *im2*)

Modulo.

The function performs modulo (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMod** (p. 449). The image functor instantiator : **HxInstantiatorMod** (p. 897).

```
13 {
14     HxString fname("HxMod");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.signature().imageDimensionality() != im2.signature().imageDimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if (im1.signature().pixelDimensionality() != im2.signature().pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35         return HxImageRep();
36     }
37     if (im1.signature().pixelDimensionality() != 1)
38     {
39         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
40         return HxImageRep();
41     }
```

```
42
43      if (im1.signature().pixelType() != im2.signature().pixelType())
44      {
45          HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46          return HxImageRep();
47      }
48      if (im1.signature().pixelType() != INT_VALUE)
49      {
50          HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer values", HxGlobalE
51          return HxImageRep();
52      }
53
54      if (im1.sizes().x() != im2.sizes().x())
55      {
56          HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
57          return HxImageRep();
58      }
59      if (im1.sizes().y() != im2.sizes().y())
60      {
61          HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
62          return HxImageRep();
63      }
64      if (im1.signature().imageDimensionality() > 2)
65      {
66          if (im1.sizes().z() != im2.sizes().z())
67          {
68              HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
69              return HxImageRep();
70          }
71      }
72
73      return im1.binaryPixOp(im2, "mod");
74  }
```

7.180 HxModVal.h File Reference

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

Functions

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

  Modulo.

7.180.1 Detailed Description

7.180.2 Function Documentation

7.180.2.1 HxImageRep L_HXIMAGEREP HxModVal (HxImageRep *im*, HxValue *val*)

Modulo.

The function performs modulo (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMod** (p. 449). The image functor instantiator : **HxInstantiatorModV** (p. 897).

```
13 {
14     HxString fname("HxModVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     if (im.signature().pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
25         return HxImageRep();
26     }
27
28     if (im.signature().pixelType() != INT_VALUE)
29     {
30         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
31         return HxImageRep();
32     }
33
34     if (val.tag() != HxValue::SI)
35     {
36         HxGlobalError::instance()->reportError(fname, "only scalar integer value supported", HxGlobalError::HX_GE_INV
37         return HxImageRep();
38     }
39
40     return im.binaryPixOp(val, "mod");
41 }
```

7.181 HxMorphologicalContour.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMorphologicalContour** (**HxImageRep** im, **HxSF** sf1)

7.181.1 Detailed Description

7.182 HxMorphologicalGradient.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMorphologicalGradient** (**HxImageRep** im, **HxSF** sf1)

  use the same SF for erosion and dilation.

7.182.1 Detailed Description

7.182.2 Function Documentation

7.182.2.1 HxImageRep L_HXIMAGEREP HxMorphologicalGradient (HxImageRep *im*, HxSF *sf1*)

use the same SF for erosion and dilation.

formula: return dilation(im,sf) - erosion(im,sf)

```
17 {
18     return HxSub( HxDilation(im, sf1), HxErosion(im, sf1) );
19 }
```

7.183 HxMorphologicalGradient2.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxMorphologicalGradient2** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)

  use the different SF for dilation (sf1) and erosion (sf2).

7.183.1 Detailed Description

7.183.2 Function Documentation

7.183.2.1 HxImageRep L_HXIMAGEREP HxMorphologicalGradient2 (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

use the different SF for dilation (sf1) and erosion (sf2).

formula: return dilation(im,sf1) - erosion(im,sf2)

```
17 {
18     return HxSub( HxDilation(im, sf1), HxErosion(im, sf2) );
19 }
```

7.184 HxMul.h File Reference

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

Functions

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

  Multiplication.

7.184.1 Detailed Description

7.184.2 Function Documentation

7.184.2.1 HxImageRep L_HXIMAGEREP HxMul (HxImageRep *im1*, HxImageRep *im2*)

Multiplication.

The function performs multiplication (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMul** (p. 450). The image functor instantiator : **HxInstantiatorMul** (p. 898).

```
16 {
17     HxString fname("HxMul");
18
19     if (im1.isNull())
20     {
21         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
22         return HxImageRep();
23     }
24     if (im2.isNull())
25     {
26         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
27         return HxImageRep();
28     }
29
30     if (im1.dimensionality() != im2.dimensionality())
31     {
32         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
33         return HxImageRep();
34     }
35     if ((im1.pixelDimensionality() != im2.pixelDimensionality()) &&
36         (im1.pixelDimensionality() != 1) &&
37         (im2.pixelDimensionality() != 1))
38     {
39         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
40         return HxImageRep();
41     }
42
43     if (im1.sizes().x() != im2.sizes().x())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.sizes().y() != im2.sizes().y())
49     {
50         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
51         return HxImageRep();
52     }
53     if (im1.dimensionality() > 2)
54     {
55         if (im1.sizes().z() != im2.sizes().z())
56         {
57             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
58             return HxImageRep();
59         }
60     }
61
62     // in case of byte, unsigned: generate warnings in case of potentially dangerous
63     // situations.
64     // Check if image is byte.
```

```
65     if (((im1.signature().pixelType() == INT_VALUE) &&
66          (im1.signature().pixelPrecision() == 8)) ||
67         ((im2.signature().pixelType() == INT_VALUE) &&
68          (im2.signature().pixelPrecision() == 8)))
69     {
70         if ((im1.pixelDimensionality() == 1) && (im1.pixelDimensionality() == 1))
71         {
72             if ((HxPixMax(im1).HxScalarIntValue() *
73                  HxPixMax(im2).HxScalarIntValue()) > HxScalarInt(255))
74             {
75                 HxGlobalError::instance()->reportWarning(fname,
76                     im1.name()+HxString(" ")+im2.name(),
77                     "possible overflow due to byte precision",
78                     HxGlobalError::HX_GW_OVERFLOW);
79             }
80         }
81         else if ((HxPixMax(HxUnaryMax(im1)).HxScalarIntValue() *
82                   HxPixMax(HxUnaryMax(im2)).HxScalarIntValue()) > HxScalarInt(255))
83         {
84                 HxGlobalError::instance()->reportWarning(fname,
85                     im1.name()+HxString(" ")+im2.name(),
86                     "possible overflow due to byte precision",
87                     HxGlobalError::HX_GW_OVERFLOW);
88         }
89     }
90
91     return im1.binaryPixOp(im2, "mul");
92 }
```

7.185 HxMulVal.h File Reference

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

Functions

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

  Multiplication.

7.185.1 Detailed Description

7.185.2 Function Documentation

7.185.2.1 HxImageRep L_HXIMAGEREP HxMulVal (HxImageRep *im*, HxValue *val*)

Multiplication.

The function performs multiplication (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMul** (p. 450). The image functor instantiator : **HxInstantiatorMulV** (p. 899).

```
13 {
14     HxString fname("HxMulVal");
15
```

```
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if ((valdim != 1) && (im.signature().pixelDimensionality() != valdim))
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "mul");
43 }
```

7.186 HxNegate.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNegate** (**HxImageRep** im)

  Negation.

7.186.1 Detailed Description

7.186.2 Function Documentation

7.186.2.1 HxImageRep L_HXIMAGEREP HxNegate (HxImageRep *im*)

Negation.

The function computes the negation (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoNegate** (p. 1232). The image functor instantiator : **HxInstantiatorNegate** (p. 900).

```
13 {
14     HxString fname("HxNegate");
15
```

```
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("negate");
23 }
```

7.187 HxNonMaxSuppressionGradDir.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNonMaxSuppressionGradDir** (**HxImageRep** img)

  Non maxima suppression in the direction of the gradient.

7.187.1 Detailed Description

7.187.2 Function Documentation

7.187.2.1 HxImageRep L_HXIMAGEREP HxNonMaxSuppressionGradDir (HxImageRep *img*)

Non maxima suppression in the direction of the gradient.

Implementation specifics : The neighbourhood functor : **HxNgbNonMaxSuppression2d** (p. 1056). The image functor instantiator : **HxInstNgbNonMaxSuppression2d_c** (p. **??**).

```
13 {
14     HxString fname("HxNonMaxSuppressionGradDir");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return img.neighbourhoodOp("nonMaxSuppression");
23 }
```

7.188 HxNorm1.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNorm1** (**HxImageRep** im)

  L1 norm (city block distance).

7.188.1 Detailed Description

7.188.2 Function Documentation

7.188.2.1 HxImageRep L_HXIMAGEREP HxNorm1 (HxImageRep *im*)

L1 norm (city block distance).

The function computes the L1 norm (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoNorm1** (p. 1233). The image functor instantiator : **HxInstantiatorNorm1** (p. 901).

```
13 {
14     return im.unaryPixOp("norm1");
15 }
```

7.189 HxNorm2.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNorm2** (**HxImageRep** im)

  L2 norm (Euclidain norm).

7.189.1 Detailed Description

7.189.2 Function Documentation

7.189.2.1 HxImageRep L_HXIMAGEREP HxNorm2 (HxImageRep *im*)

L2 norm (Euclidain norm).

The function computes the L2 norm (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoNorm2** (p. 1235). The image functor instantiator : **HxInstantiatorNorm2** (p. 901).

```
13 {
14     return im.unaryPixOp("norm2");
15 }
```

7.190 HxNorm2Sqr.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNorm2Sqr** (**HxImageRep** im)

  Squared L2 norm (squared Euclidain norm).

7.190.1 Detailed Description

7.190.2 Function Documentation

7.190.2.1 HxImageRep L_HXIMAGEREP HxNorm2Sqr (HxImageRep *im*)

Squared L2 norm (squared Euclidain norm).

The function computes the squared L2 norm (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoNorm2Sqr** (p. 1236). The image functor instantiator : **HxInstantiatorNorm2Sqr** (p. 902).

```
13 {
14     return im.unaryPixOp("norm2sqr");
15 }
```

7.191 HxNormalizedCorrelation.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNormalizedCorrelation** (**HxImageRep** img, **HxImageRep** kernel)

  Normalized (cross) correlation.

7.191.1 Detailed Description

7.191.2 Function Documentation

7.191.2.1 HxImageRep L_HXIMAGEREP HxNormalizedCorrelation (HxImageRep *img*, HxImageRep *kernel*)

Normalized (cross) correlation.

Implementation specifics : The neighbourhood functor : **HxKerNgbNormCorrelation** (p. 973). The image functor instantiator : **HxInstKerNgb2dNormCorrelation_c** (p. **??**).

```
13 {
14     HxString fname("HxNormalizedCorrelation");
15
16     if (img.isNull())
```

```
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (kernel.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, kernel.name(), "null kernel", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26     if (img.dimensionality() != kernel.dimensionality())
27     {
28         HxGlobalError::instance()->reportError(fname, "kernel and image dimensionality do not match", HxGlobalError:
29         return HxImageRep();
30     }
31     if (kernel.pixelDimensionality() != img.pixelDimensionality())
32     {
33         HxGlobalError::instance()->reportError(fname, "kernel and image pixel dimensionality do not match", HxGlobalE
34         return HxImageRep();
35     }
36
37     return img.neighbourhoodOp(kernel, "normalizedCorrelation");
38 }
```

7.192 HxNormInf.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxNormInf** (**HxImageRep** im)

  L infinity norm (chessboard).

7.192.1 Detailed Description

7.192.2 Function Documentation

7.192.2.1 HxImageRep L_HXIMAGEREP HxNormInf (HxImageRep *im*)

L infinity norm (chessboard).

The function computes the L infinity norm (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoNormInf** (p. 1238). The image functor instantiator : **HxInstantiatorNormInf** (p. 903).

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

7.193 HxNotEqual.h File Reference

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

Functions

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

  Not equal.

7.193.1 Detailed Description

7.193.2 Function Documentation

7.193.2.1 HxImageRep L_HXIMAGEREP HxNotEqual (HxImageRep *im1*, HxImageRep *im2*)

Not equal.

The function performs not equal (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoNotEqual** (p. 453). The image functor instantiator : **HxInstantiatorNotEqual** (p. 903).

```
13 {
14     HxString fname("HxNotEqual");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32     if (im1.pixelDimensionality() != im2.pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
35         return HxImageRep();
36     }
37 
38     if (im1.sizes().x() != im2.sizes().x())
39     {
40         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
41         return HxImageRep();
42     }
43     if (im1.sizes().y() != im2.sizes().y())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.dimensionality() > 2)
49     {
50         if (im1.sizes().z() != im2.sizes().z())
51         {
52             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
53             return HxImageRep();
```

```
54         }
55     }
56
57     return im1.binaryPixOp(im2, "notEqual");
58 }
```

7.194 HxNotEqualVal.h File Reference

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

Functions

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

  Not equal.

7.194.1 Detailed Description

7.194.2 Function Documentation

7.194.2.1 HxImageRep L_HXIMAGEREP HxNotEqualVal (HxImageRep *im*, HxValue *val*)

Not equal.

The function performs not equal (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoNotEqual** (p. 453). The image functor instantiator : **HxInstantiatorNotEqualV** (p. 904).

```
13 {
14     HxString fname("HxNotEqualVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if (im.signature().pixelDimensionality() != valdim)
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
```

```
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "notEqual");
43 }
```

7.195 HxOpening.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOpening** (**HxImageRep** im, **HxSF** sf)

7.195.1 Detailed Description

7.196 HxOpeningByReconstruction.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOpeningByReconstruction** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)

7.196.1 Detailed Description

7.197 HxOpeningByReconstructionTopHat.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOpeningByReconstructionTopHat** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)

  function y=mmopenrecth_equ(f, bero, bc) y = mmsubm(f,mmopenrec(f, bero, bc));.

7.197.1 Detailed Description

7.197.2 Function Documentation

7.197.2.1 HxImageRep L_HXIMAGEREP HxOpeningByReconstructionTopHat (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

function y=mmopenrecth_equ(f, bero, bc) y = mmsubm(f,mmopenrec(f, bero, bc));.

```
24 {
25 //  return HxSubSat(im, HxOpeningByReconstruction(im, sf1,sf2));
26     return HxSub(im, HxOpeningByReconstruction(im, sf1,sf2));
27 }
```

7.198 HxOpeningTopHat.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxOpeningTopHat** (**HxImageRep** im, **HxSF** sf)

  res = im - opening(im,sf).

7.198.1 Detailed Description

7.198.2 Function Documentation

7.198.2.1 HxImageRep L_HXIMAGEREP HxOpeningTopHat (HxImageRep *im*, HxSF *sf*)

res = im - opening(im,sf).

```
15 {
16     return HxSub(im, HxOpening(im,sf));
17 }
```

7.199 HxOpponentColor.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxOpponentColor** (**HxImageRep** im)

7.199.1 Detailed Description

7.200 HxOpticalFlow.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxOFOneScale** (**HxImageRep** Ix, **HxImageRep** Iy, **HxImage-Rep** It)

implement Lucas and Kanade's method for optical flow detection then go for Bergen - multiscale the output is a HxVec2DDouble image where the displacement x and y values are exported.

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

7.200.1 Detailed Description

7.200.2 Function Documentation

7.200.2.1 HxImageRep L_HXIMAGEREP HxOFOneScale (HxImageRep *Ix*, HxImageRep *Iy*, HxImageRep *It*)

implement Lucas and Kanade's method for optical flow detection then go for Bergen - multiscale the output is a HxVec2DDouble image where the displacement x and y values are exported.

```
15 {
16     return Ix.neighbourhoodOpExtra2("opticalFlow",Iy,It);
17 }
```

7.201 HxOpticalFlowMultiScale.h File Reference

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

Functions

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

  implement Lucas and Kanade's method for optical flow detection then go for Bergen - multiscale the output is a HxVec2DDouble image where the displacement x and y values are exported.

7.201.1 Detailed Description

7.201.2 Function Documentation

7.201.2.1 HxImageRep L_HXIMAGEREP HxOpticalFlowMultiScale (HxImageRep *im1*, HxImageRep *im2*)

implement Lucas and Kanade's method for optical flow detection then go for Bergen - multiscale the output is a HxVec2DDouble image where the displacement x and y values are exported.

```
17 {
18     //to compute the image derivatives
19     //for x and y use the coefficients: 1/12(-1,8,0,-8,1)
20     int pixelDimensionality=1;
21     int dimensions=2;
22     HxSizes sizes(5,1,1);
23     //the data pointer for Hx should have 3 shorts per pixels?
24     double data[]={ -1.0/12,  8.0/12,  0, -8.0/12, 1.0/12 };
25
26     HxImageRep kernel = HxMakeFromDoubleData(pixelDimensionality, dimensions, sizes,    data);
```

```
27
28
29      //create the pyramid images
30      double sigma = 1.5;
31      HxImageRep im1_1=HxGauss(im1,sigma,HxImageRep::ARITH_PREC);
32      HxImageRep im2_1=HxGauss(im2,sigma,HxImageRep::ARITH_PREC);
33      im1_1 = HxScale(im1_1, 0.5, 0.5, 1, NEAREST, 1); //NEAREST
34      im2_1 = HxScale(im2_1, 0.5, 0.5, 1, NEAREST, 1);
35
36
37 //   sigma = 2.0;
38      HxImageRep im1_2=HxGauss(im1_1,sigma,HxImageRep::ARITH_PREC);
39      HxImageRep im2_2=HxGauss(im2_1,sigma,HxImageRep::ARITH_PREC);
40 //   im1_2 = HxScale(im1_2, 0.25, 0.25, 1, LINEAR, 1);
41 //   im2_2 = HxScale(im2_2, 0.25, 0.25, 1, LINEAR, 1);
42      im1_2 = HxScale(im1_2, 0.5, 0.5, 1, NEAREST, 1);
43      im2_2 = HxScale(im2_2, 0.5, 0.5, 1, NEAREST, 1);
44
45
46      //you have to start with the smallest image (top of the pyramid)
47
48      HxImageRep ix,iy,it; //derivatives on x, y and time
49      HxImageRep ix_1,iy_1,it_1; //derivatives on x, y and time
50      HxImageRep ix_2,iy_2,it_2; //derivatives on x, y and time
51
52      ix_2 = im1_2.generalizedConvolutionK1d(1, kernel,"mul", "addAssign", HxImageRep::ARITH_PREC);
53      iy_2 = im1_2.generalizedConvolutionK1d(2, kernel,"mul", "addAssign", HxImageRep::ARITH_PREC);
54      it_2 = HxSub(im2_2,im1_2);
55
56
57      HxImageRep res2 = HxOFOneScale(ix_2,iy_2,it_2);
58
59      res2 = HxScale(res2, 2, 2, 1, LINEAR, 1);
60      res2 = HxMulVal(res2,2);
61
62      ix_1 = im1_1.generalizedConvolutionK1d(1, kernel,"mul", "addAssign", HxImageRep::ARITH_PREC);
63      iy_1 = im1_1.generalizedConvolutionK1d(2, kernel,"mul", "addAssign", HxImageRep::ARITH_PREC);
64      //estimate now the image at time t+1 by motion in smallar image
65      it_1 = HxSub(im2_2,HxAdd(im1_1,res2));
66
67      HxImageRep res1 = HxOFOneScale(ix_1,iy_1,it_1);
68
69      res1 = HxScale(res1, 2, 2, 1, LINEAR, 1);
70      res1 = HxMulVal(res1,2);
71
72      ix = im1.generalizedConvolutionK1d(1, kernel,"mul", "addAssign", HxImageRep::ARITH_PREC);
73      iy = im1.generalizedConvolutionK1d(2, kernel,"mul", "addAssign", HxImageRep::ARITH_PREC);
74      //estimate now the image at time t+1 by motion in smallar image
75      it = HxSub(im2,HxAdd(im1,res1));
76
77      /*
78      HxWriteFile(ix,"ix.tif");
79      HxWriteFile(iy,"iy.tif");
80      HxWriteFile(it,"it.tif");
81      */
82
83      HxImageRep res = HxOFOneScale(ix,iy,it);
84
85      //export now the two components to vizualize with Hieu's program
86 /*
87      HxImageRep u = HxProjectRange(res,1);
88      HxImageRep v = HxProjectRange(res,2);
89
90 //   u = HxAbs(u);
91 //   v = HxAbs(v);
```

```
92      double umin = HxPixMax(u).HxScalarDoubleValue().x();
93      double vmin = HxPixMax(v).HxScalarDoubleValue().x();
94
95      u = HxAddVal(u,fabs(umin));
96      v = HxAddVal(v,fabs(vmin));
97
98      double umax = HxPixMax(HxAbs(u)).HxScalarDoubleValue().x();
99      double vmax = HxPixMax(HxAbs(v)).HxScalarDoubleValue().x();
100      HxWriteFile(HxImageAsByte(HxMulVal(u,255.0/umax)),"u1.tif");
101      HxWriteFile(HxImageAsByte(HxMulVal(v,255.0/vmax)),"v1.tif");
102
103 */
104     return res;
105 }
```

7.202 HxOr.h File Reference

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

Functions

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

  Or.

7.202.1 Detailed Description

7.202.2 Function Documentation

7.202.2.1 HxImageRep L_HXIMAGEREP HxOr (HxImageRep *im1*, HxImageRep *im2*)

Or.

The function performs or (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoOr** (p. 455). The image functor instantiator : **HxInstantiatorOr** (p. 904).

```
13 {
14      HxString fname("HxOr");
15
16      if (im1.isNull())
17      {
18          HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19          return HxImageRep();
20      }
21      if (im2.isNull())
22      {
23          HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24          return HxImageRep();
25      }
26
27      if (im1.dimensionality() != im2.dimensionality())
28      {
29          HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
```

```
30          return HxImageRep();
31      }
32      if (im1.pixelDimensionality() != im2.pixelDimensionality())
33      {
34          HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35          return HxImageRep();
36      }
37      if (im1.pixelDimensionality() != 1)
38      {
39          HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
40          return HxImageRep();
41      }
42
43      if (im1.pixelType() != im2.pixelType())
44      {
45          HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46          return HxImageRep();
47      }
48      if (im1.pixelType() != INT_VALUE)
49      {
50          HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer values", HxGlobalE
51          return HxImageRep();
52      }
53
54      if (im1.pixelPrecision() != im2.pixelPrecision())
55      {
56          HxGlobalError::instance()->reportError(fname, "unequal pixel precisions", HxGlobalError::HX_GE_UNEQUAL_IMAGES
57          return HxImageRep();
58      }
59
60      if (im1.sizes().x() != im2.sizes().x())
61      {
62          HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
63          return HxImageRep();
64      }
65      if (im1.sizes().y() != im2.sizes().y())
66      {
67          HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
68          return HxImageRep();
69      }
70      if (im1.dimensionality() > 2)
71      {
72          if (im1.sizes().z() != im2.sizes().z())
73          {
74              HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
75              return HxImageRep();
76          }
77      }
78
79      return im1.binaryPixOp(im2, "or");
80  }
```

7.203 HxOrVal.h File Reference

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

Functions

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

  Or.

7.203.1 Detailed Description

7.203.2 Function Documentation

7.203.2.1 HxImageRep L_HXIMAGEREP HxOrVal (HxImageRep *im*, HxValue *val*)

Or.

The function performs or (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoOr** (p. 455). The image functor instantiator : **HxInstantiatorOrV** (p. 905).

```
13 {
14     HxString fname("HxOrVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     if (im.signature().pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
25         return HxImageRep();
26     }
27
28     if (im.signature().pixelType() != INT_VALUE)
29     {
30         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
31         return HxImageRep();
32     }
33
34     if (val.tag() != HxValue::SI)
35     {
36         HxGlobalError::instance()->reportError(fname, "only scalar integer value supported", HxGlobalError::HX_GE_INV
37         return HxImageRep();
38     }
39
40
41     return im.binaryPixOp(val, "or");
42 }
```

7.204 HxParabolicDilation.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxParabolicDilation** (**HxImageRep** img, double rho, double accuracy=3.0)

  Parabolic dilation.

7.204.1 Detailed Description

7.204.2 Function Documentation

7.204.2.1 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     HxString fname("HxParabolicDilation");
19
20     if (img.isNull())
21     {
22         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
23         return HxImageRep();
24     }
25
26     if (rho <= 0.0)
27     {
28         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of rho", HxGlobalError::HX_GE_INVAL
29         return HxImageRep();
30     }
31     if (accuracy < 0.0)
32     {
33         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of accuracy", HxGlobalError::HX_GE_
34         return HxImageRep();
35     }
36
37     int minSize = HxImageMinSize(img);
38     HxImageRep parabola = HxMakeParabola1d(rho, accuracy, minSize);
39     return img.genConvSeparated(
40         parabola, "add", "maxAssign", HxImageRep::ARITH_PREC);
41 }
```

7.205 HxParabolicErosion.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxParabolicErosion** (**HxImageRep** img, double rho, double accuracy=3.0)

  Parabolic erosion.

7.205.1 Detailed Description

7.205.2 Function Documentation

7.205.2.1 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     HxString fname("HxParabolicErosion");
19
20     if (img.isNull())
21     {
22         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
23         return HxImageRep();
24     }
25
26     if (rho <= 0.0)
27     {
28         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of rho", HxGlobalError::HX_GE_INVAL
29         return HxImageRep();
30     }
31     if (accuracy < 0.0)
32     {
33         HxGlobalError::instance()->reportError(fname, img.name(), "invalid value of accuracy", HxGlobalError::HX_GE_
34         return HxImageRep();
35     }
36
37     int minSize = HxImageMinSize(img);
38     HxImageRep parabola = HxMakeParabola1d(-rho, accuracy, minSize);
39     return img.genConvSeparated(
40         parabola, "add", "minAssign", HxImageRep::ARITH_PREC);
41 }
```

7.206 HxPeakRemoval.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxPeakRemoval** (**HxImageRep** im, int conn, int minarea)

7.206.1 Detailed Description

7.207 HxPercentile.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxPercentile** (**HxImageRep** im, int neighSize, double perc)
  Percentile filter.

7.207.1 Detailed Description

7.207.2 Function Documentation

7.207.2.1 HxImageRep L_HXIMAGEREP HxPercentile (HxImageRep *im*, int *neighSize*, double *perc*)

Percentile filter.

Implementation specifics : The neighbourhood functor : **HxNgbPercentile2d** (p. 1060). The image functor instantiator : **HxInstNgbPercentile2d_c** (p. **??**).

```
14 {
15     HxString fname("HxPercentile");
16
17     if (im.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22
23     if (im.dimensionality() != 2)
24     {
25         HxGlobalError::instance()->reportError(fname, im.name(), "Only valid for 2D images", HxGlobalError::HX_GE_IN
26         return HxImageRep();
27     }
28     if (neighSize < 1)
29     {
30         HxGlobalError::instance()->reportError(fname, "neighbourhoud too small", HxGlobalError::HX_GE_INVALID);
31         return HxImageRep();
32     }
33     if ((neighSize > im.dimensionSize(1)) || (neighSize > im.dimensionSize(2)))
34     {
35         HxGlobalError::instance()->reportError(fname, "neighbourhoud too large", HxGlobalError::HX_GE_INVALID);
36         return HxImageRep();
37     }
38     if ((perc < 0) || (perc > 1))
39     {
40         HxGlobalError::instance()->reportError(fname, "percentile in [0:1]", HxGlobalError::HX_GE_INVALID);
41         return HxImageRep();
42     }
43
44     HxTagList tags;
45     HxAddTag(tags, "size", neighSize);
46     HxAddTag(tags, "percentile", perc);
47
48     return im.neighbourhoodOp("percentile", tags);
49 }
```

7.208 HxPixInf.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxPixInf** (**HxImageRep** im)

  Pixel infimum.

7.208.1 Detailed Description

7.208.2 Function Documentation

7.208.2.1 HxValue L_HXIMAGEREP HxPixInf (HxImageRep *im*)

Pixel infimum.

The function computes the infimum (see **Pixels** (p. 3)) of all pixels in the input image via a reduce operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoInfAssign** (p. 437). The image functor instantiator : **HxInstantiatorInfReduce** (p. 887).

```
13 {
14     HxString fname("HxPixInf");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxValue();
20     }
21
22     return im.reduceOp("infAssign");
23 }
```

7.209 HxPixMax.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxPixMax** (**HxImageRep** im)

  Pixel maximum.

7.209.1 Detailed Description

7.209.2 Function Documentation

7.209.2.1 HxValue L_HXIMAGEREP HxPixMax (HxImageRep *im*)

Pixel maximum.

The function computes the maximum (see **Pixels** (p. 3)) of all pixels in the input image via a reduce operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMaxAssign** (p. 444). The image functor instantiator : **HxInstantiatorMaxReduce** (p. 894).

```
13 {
14     HxString fname("HxPixMax");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxValue();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, im.name(), "Operation only valid on scalar images", HxGlobalEr
24         return HxValue();
25     }
26     return im.reduceOp("maxAssign");
27 }
```

7.210 HxPixMin.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxPixMin** (**HxImageRep** im)

  Pixel minimum.

7.210.1 Detailed Description

7.210.2 Function Documentation

7.210.2.1 HxValue L_HXIMAGEREP HxPixMin (HxImageRep *im*)

Pixel minimum.

The function computes the minimum (see **Pixels** (p. 3)) of all pixels in the input image via a reduce operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMinAssign** (p. 447). The image functor instantiator : **HxInstantiatorMinReduce** (p. 895).

```
13 {
14     HxString fname("HxPixMin");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxValue();
20     }
21     if (im.pixelDimensionality() != 1)
22     {
23         HxGlobalError::instance()->reportError(fname, im.name(), "Operation only valid on scalar images", HxGlobalEr
24         return HxValue();
```

```
25     }
26
27     return im.reduceOp("minAssign");
28 }
```

7.211 HxPixProduct.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxPixProduct** (**HxImageRep** im)

  Pixel product.

7.211.1 Detailed Description

7.211.2 Function Documentation

7.211.2.1 HxValue L_HXIMAGEREP HxPixProduct (HxImageRep *im*)

Pixel product.

The function computes the product (see **Pixels** (p. 3)) of all pixels in the input image via a reduce operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoMulAssign** (p. 452). The image functor instantiator : **HxInstantiatorMulReduce** (p. 899).

```
13 {
14     HxString fname("HxPixProduct");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxValue();
20     }
21
22     return im.reduceOp("mulAssign");
23 }
```

7.212 HxPixSum.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxPixSum** (**HxImageRep** im)

  Pixel sum.

7.212.1 Detailed Description

7.212.2 Function Documentation

7.212.2.1 HxValue L_HXIMAGEREP HxPixSum (HxImageRep *im*)

Pixel sum.

The function computes the sum (see **Pixels** (p. 3)) of all pixels in the input image via a reduce operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoAddAssign** (p. 419). The image functor instantiator : **HxInstantiatorAddReduce** (p. 866).

```
13 {
14     HxString fname("HxPixSum");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxValue();
20     }
21
22     return im.reduceOp("addAssign");
23 }
```

7.213 HxPixSup.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxPixSup** (**HxImageRep** im)

  Pixel supremum.

7.213.1 Detailed Description

7.213.2 Function Documentation

7.213.2.1 HxValue L_HXIMAGEREP HxPixSup (HxImageRep *im*)

Pixel supremum.

The function computes the supremum (see **Pixels** (p. 3)) of all pixels in the input image via a reduce operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoSupAssign** (p. 467). The image functor instantiator : **HxInstantiatorSupReduce** (p. 921).

```
13 {
14     HxString fname("HxPixSup");
15
16     if (im.isNull())
```

```
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxValue();
20     }
21
22     return im.reduceOp("supAssign");
23 }
```

7.214 HxPoint.h File Reference

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

Typedefs

- typedef **HxVec3Double HxPoint**
  Definition of a point in 3D space.

7.214.1 Detailed Description

7.214.2 Typedef Documentation

7.214.2.1 typedef HxVec3Double HxPoint

Definition of a point in 3D space.

Uses floating point (double) coordinates.

7.215 HxPointInt.h File Reference

```
#include "HxVec3Int.h"
```

Typedefs

- typedef **HxVec3Int HxPointInt**
  Definition of a point in 3D space using integer coordinates.

7.215.1 Detailed Description

7.215.2 Typedef Documentation

7.215.2.1 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.

7.216 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. 1073).
- typedef HxPointList::const_iterator **HxPointListConstIter**
  Const iterator for **HxPointList** (p. 1073).
- typedef **HxPointList::back_insert_iterator HxPointListBackInserter**
  Back inserter for **HxPointList** (p. 1073).

7.216.1 Detailed Description

7.216.2 Typedef Documentation

7.216.2.1 typedef HxPointList::iterator HxPointListIter

Iterator for **HxPointList** (p. 1073).

7.216.2.2 typedef HxPointList::const_iterator HxPointListConstIter

Const iterator for **HxPointList** (p. 1073).

7.216.2.3 typedef HxPointList::back_insert_iterator HxPointListBackInserter

Back inserter for **HxPointList** (p. 1073).

7.217 HxPow.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxPow** (**HxImageRep** im1, **HxImageRep** im2)
  Power.

7.217.1 Detailed Description

7.217.2 Function Documentation

7.217.2.1 HxImageRep L_HXIMAGEREP HxPow (HxImageRep *im1*, HxImageRep *im2*)

Power.

The function performs power (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoPow** (p. 456). The image functor instantiator : **HxInstantiatorPow** (p. 906).

```
13 {
14     HxString fname("HxPow");
15 
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im1.dimensionality() != im2.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL
30         return HxImageRep();
31     }
32     if ((im1.pixelDimensionality() != im2.pixelDimensionality()) &&
33         (im2.pixelDimensionality() != 1))
34     {
35         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL
36         return HxImageRep();
37     }
38 
39     if (im1.sizes().x() != im2.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im1.sizes().y() != im2.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (im1.dimensionality() > 2)
50     {
51         if (im1.sizes().z() != im2.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGE
```

```
54              return HxImageRep();
55          }
56      }
57
58      // in case of byte, unsigned: generate warnings in case of potentially dangerous
59      // situations.
60      // Check if image is byte.
61      if (((im1.signature().pixelType() == INT_VALUE) &&
62           (im1.signature().pixelPrecision() == 8)) ||
63          ((im2.signature().pixelType() == INT_VALUE) &&
64           (im2.signature().pixelPrecision() == 8)))
65      {
66          if ((im1.pixelDimensionality() == 1) && (im1.pixelDimensionality() == 1))
67          {
68              if (pow(HxPixMax(im1).HxScalarIntValue().x(),
69                   HxPixMax(im2).HxScalarIntValue().x()) > 255)
70              {
71                  HxGlobalError::instance()->reportWarning(fname,
72                      im1.name()+HxString(" ")+im2.name(),
73                      "possible overflow due to byte precision",
74                      HxGlobalError::HX_GW_OVERFLOW);
75              }
76          }
77          else if (pow(HxPixMax(HxUnaryMax(im1)).HxScalarIntValue().x(),
78                    HxPixMax(HxUnaryMax(im2)).HxScalarIntValue().x()) > 255)
79          {
80                  HxGlobalError::instance()->reportWarning(fname,
81                      im1.name()+HxString(" ")+im2.name(),
82                      "possible overflow due to byte precision",
83                      HxGlobalError::HX_GW_OVERFLOW);
84          }
85      }
86
87      return im1.binaryPixOp(im2, "pow");
88  }
```

7.218 HxPowVal.h File Reference

`#include "HxImageRep.h"`

Functions

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

  Power.

7.218.1 Detailed Description

7.218.2 Function Documentation

7.218.2.1 HxImageRep L_HXIMAGEREP HxPowVal (HxImageRep *im*, HxValue *val*)

Power.

The function performs power (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoPow** (p. 456). The image functor instantiator : **HxInstantiatorPowV** (p. 906).

```
13 {
14     HxString fname("HxPowVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if ((valdim != 1) && (im.signature().pixelDimensionality() != valdim))
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "pow");
43 }
```

7.219 HxProjectRange.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxProjectRange** (**HxImageRep** im, int dimension)

  Projection of the pixel range.

7.219.1 Detailed Description

7.219.2 Function Documentation

7.219.2.1 HxImageRep L_HXIMAGEREP HxProjectRange (HxImageRep *im*, int *dimension*)

Projection of the pixel range.

The function computes the projection (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)). Dimension starts at 1.

```
13 {
14     HxString fname("HxProjectRange");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if ((im.pixelDimensionality() != 2) &&
22         (im.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "Operation is only valid for Vector images", HxGlobalError::HX_
25         return HxImageRep();
26     }
27     if (dimension < 1)
28     {
29         HxGlobalError::instance()->reportError(fname, "Dimension should be greater than zero", HxGlobalError::HX_GE_
30         return HxImageRep();
31     }
32     if (dimension > im.pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "Dimension should be less than pixel dimensionality", HxGlobal
35         return HxImageRep();
36     }
37
38     return im.projectRange(dimension);
39 }
```

7.220 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)

  Recursive Gaussian.

7.220.1 Detailed Description

7.220.2 Function Documentation

7.220.2.1 HxImageRep L_HXIMAGEREP HxRecGauss (HxImageRep *im*, double *sx*, double *sy*, int *dx* = 0, int *dy* = 0, int *recurOrder* = 3)

Recursive Gaussian.

```
65 {
66     GaussIIR gx(sx, dx, recurOrder);
67     GaussIIR gy(sy, dy, recurOrder);
68     HxImageRep fx = HxImageFactory::instance().fromGenerator(HXIMAGESIG2DDOUBLE, &gx);
69     HxImageRep fy = HxImageFactory::instance().fromGenerator(HXIMAGESIG2DDOUBLE, &gy);
70
71     HxImageRep res = im.recGenConv2dSep(fx, fy, "mul", "addAssign",
72                                         HxImageRep::ARITH_PREC);
```

```
73
74      switch (dx) {
75      case 1: {
76          FilterN fdx(-0.5,0,0.5);
77          HxImageRep hdx = HxImageFactory::instance().fromGenerator(
78                                                      HXIMAGESIG2DDOUBLE, &fdx);
79          res = res.generalizedConvolutionK1d(1,hdx, "mul", "addAssign");
80         }
81         break;
82      case 2: {
83          FilterN fdx(1,-2,1);
84          HxImageRep hdx = HxImageFactory::instance().fromGenerator(
85                                                      HXIMAGESIG2DDOUBLE, &fdx);
86          res = res.generalizedConvolutionK1d(1,hdx, "mul", "addAssign");
87         }
88         break;
89      }
90      switch (dy) {
91      case 1: {
92          FilterN fdy(-0.5,0,0.5);
93          HxImageRep hdy = HxImageFactory::instance().fromGenerator(
94                                                      HXIMAGESIG2DDOUBLE, &fdy);
95          res = res.generalizedConvolutionK1d(2,hdy, "mul", "addAssign");
96         }
97         break;
98      case 2: {
99          FilterN fdy(1,-2,1);
100          HxImageRep hdy = HxImageFactory::instance().fromGenerator(
101                                                       HXIMAGESIG2DDOUBLE, &fdy);
102          res = res.generalizedConvolutionK1d(2,hdy, "mul", "addAssign");
103         }
104         break;
105      }
106
107      return res;
108 }
```

7.221 HxReciprocal.h File Reference

`#include "HxImageRep.h"`

Functions

- **HxImageRep** L_HXIMAGEREP **HxReciprocal** (**HxImageRep** im)

  Computes the reciprocal (1/x) of the input image values.

7.221.1 Detailed Description

7.221.2 Function Documentation

7.221.2.1 HxImageRep L_HXIMAGEREP HxReciprocal (HxImageRep *im*)

Computes the reciprocal (1/x) of the input image values.

If pixel values of in are zero, the corresponding pixels in out is set to zero.

```
93 {
94     return im.unaryPixOp("reciprocal");
95 }
```

7.222 HxReflect.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxReflect** (**HxImageRep** img, int doX, int doY, int doZ=1)

  Reflection.

7.222.1 Detailed Description

7.222.2 Function Documentation

7.222.2.1 HxImageRep L_HXIMAGEREP HxReflect (HxImageRep *img*, int *doX*, int *doY*, int *doZ* = 1)

Reflection.

```
14 {
15     HxString fname("HxReflect");
16
17     if (img.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22
23     if (img.dimensionality() == 2) {
24         HxMatrix m =
25             HxMatrix::translate2d(img.dimensionSize(1)/2 - 0.5,
26                                   img.dimensionSize(2)/2 - 0.5) *
27             HxMatrix::reflect2d(doX, doY) *
28             HxMatrix::translate2d(-img.dimensionSize(1)/2 + 0.5,
29                                   -img.dimensionSize(2)/2 + 0.5);
30         return img.geometricOp2d(m, NEAREST, FORWARD, 0, HxValue(0));
31     } else {
32         HxEnvironment::instance()->errorStream()
33             << "3d reflection not implemented yet" << STD_ENDL;
34         HxEnvironment::instance()->flush();
35         return HxImageRep();
36     }
37 }
```

7.223 HxRegionalMaxima.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRegionalMaxima** (**HxImageRep** im, int conn)

  -function y=mmregmax_equ(f, bc) if (isa(f,'uint8')) k = 255; else k = 65535; fplus = mmaddm(f,1); g = mmsubm(fplus,mminfrec(f,fplus,bc)); y = mmunion(mmthreshad(g,1),mmthreshad(f,k));.

7.223.1 Detailed Description

7.223.2 Function Documentation

7.223.2.1 HxImageRep L_HXIMAGEREP HxRegionalMaxima (HxImageRep *im*, int *conn*)

-function y=mmregmax_equ(f, bc) if (isa(f,'uint8')) k = 255; else k = 65535; fplus = mmaddm(f,1); g = mmsubm(fplus,mminfrec(f,fplus,bc)); y = mmunion(mmthreshad(g,1),mmthreshad(f,k));.

```
37 {
38     HxImageRep res;
39
40
41     //in this list I put the labels representing regional maxima
42     //this list will be used in a LUT
43     //here the lut
44     vector<int> regMaxList;
45 //  regMaxList.clear();
46
47     BasicFeaturesList bfl=HxGetBlobFeatures(im, res, conn, 0);
48     HX_COUT << "bfl size=" << bfl.size() << STD_ENDL;
49     for(int ix=0;ix<bfl.size();ix++)
50     {
51         //this function works for gray images, not for color
52         int cval = bfl[ix]->val.x();
53         //now check if the the current value is the minimum among the neighbours
54             //now find the gray value of that neighbor
55         int nval;
56         int maxnval=-1;
57         for(int i=0;i<bfl[ix]->neighbors.size();i++)
58         {
59             nval=bfl[ix]->neighbors[i]->val.x();
60             if(nval>maxnval)
61                 maxnval=nval;
62         }
63         if(maxnval<cval && maxnval>=0)
64         {
65             regMaxList.push_back(bfl[ix]->label);
66 //          printf("cval=%3d, maxN=%3d\n",cval,maxnval);
67         }
68     }
69
70 //now make the LUT from the list of regional maxima
71
72
73     std::map<int,int> lut;
74     int i;
75     //lut is zero everywhere except the regional maxima position
76     //here should be from minVal to MaxVal, not from 0 to 255
77     for(i=0;i<256;i++)
78         lut[i]=0;
79
80     for(i=0;i<regMaxList.size();i++)
```

```
81         lut[regMaxList[i]] = 255;
82
83
84     res = HxLUT(res, &lut);
86     HX_COUT << "nr of regional Maxima blobs=" << regMaxList.size() << STD_ENDL;
87     return res;
88 }
```

7.224 HxRegionalMinima.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxRegionalMinima** (**HxImageRep** im, int conn)

  function y=mmregmin_equ(f, bc) fminus = mmsubm(f,1); g = mmsubm(mmsuprec(f,fminus,bc),fminus); y = mmunion(mmthreshad(g,1),mmthreshad(f,0,0));.

7.224.1 Detailed Description

7.224.2 Function Documentation

7.224.2.1 HxImageRep L_HXIMAGEREP HxRegionalMinima (HxImageRep *im*, int *conn*)

function y=mmregmin_equ(f, bc) fminus = mmsubm(f,1); g = mmsubm(mmsuprec(f,fminus,bc),fminus); y = mmunion(mmthreshad(g,1),mmthreshad(f,0,0));.

```
55 {
56     HxImageRep res;
57
58     //in this list I put the labels representing regional maxima
59     //this list will be used for LUT
60     std::vector<int> regMinList;
61     //regMinList.clear();
62
63     BasicFeaturesList bfl=HxGetBlobFeatures(im, res, conn, 0);
64     //printf("bfl size=%d\n", bfl.size());
65     for(int ix=0;ix<bfl.size();ix++)
66     {
67         //this function works for gray images, not for color
68         int cval = bfl[ix]->val.x();
69         //now check if the the current value is the minimum among the neighbours
70             //now find the gray value of that neighbor
71         int nval;
72         int minnval=INT_MAX;
73         for(int i=0;i<bfl[ix]->neighbors.size();i++)
74         {
75             nval=bfl[ix]->neighbors[i]->val.x();
76             if(nval<minnval)
77                 minnval=nval;
78         }
79         if(cval < minnval && minnval<INT_MAX)
80         {
81             regMinList.push_back(bfl[ix]->label);
82 //          printf("cval=%3d, maxN=%3d\n",cval,maxnval);
```

```
83         }
84     }
85
86 //now make the LUT from the list of regional minima
87     std::map<int,int> lut;
88     int i;
89     //lut is zero everywhere except the regional maxima position
90     //here should be from minVal to MaxVal, not from 0 to 255
91     for(i=0;i<256;i++)
92         lut[i]=0;
93
94     for(i=0;i<regMinList.size();i++)
95         lut[regMinList[i]] = 255;
96
97
98     res = HxLUT(res, &lut);
100
101
102     HX_COUT << "nr of regional Minima blobs=" << regMinList.size() << STD_ENDL;
103     return res;
104 }
```

7.225 HxRegKeyList.h File Reference

```
#include "HxRegKey.h"
#include <list>
```

Compounds

- class **HxRegKeyList**

  A list of **HxRegKeyPtr** (p. 336)*'s, that is pointers to* **HxRegKey** (p. 1097)*'s.*

Typedefs

- typedef **HxRegKey** ∗ **HxRegKeyPtr**

  Definition of a pointer to an **HxRegKey** (p. 1097).

- typedef HxRegKeyList::iterator **HxRegKeyListIter**

  Iterator for **HxRegKeyList** (p. 1104).

- typedef HxRegKeyList::const_iterator **HxRegKeyListConstIter**

  Const iterator for **HxRegKeyList** (p. 1104).

- typedef **HxRegKeyList::back_insert_iterator HxRegKeyListBackInserter**

  Back inserter for **HxPointList** (p. 1073).

7.225.1 Detailed Description

7.225.2 Typedef Documentation

7.225.2.1 typedef HxRegKey* HxRegKeyPtr

Definition of a pointer to an **HxRegKey** (p. 1097).

7.225.2.2 typedef HxRegKeyList::iterator HxRegKeyListIter

Iterator for **HxRegKeyList** (p. 1104).

7.225.2.3 typedef HxRegKeyList::const_iterator HxRegKeyListConstIter

Const iterator for **HxRegKeyList** (p. 1104).

7.225.2.4 typedef HxRegKeyList::back_insert_iterator HxRegKeyListBackInserter

Back inserter for **HxPointList** (p. 1073).

7.226 HxRegValueList.h File Reference

```
#include "HxRegValue.h"
#include <list>
```

Compounds

- class **HxRegValueList**
  A list of HxRegValuePtr's, that is pointers to **HxRegValue** (p. 1104)*'s.*

Typedefs

- typedef **HxRegValue** * **HxRegValuePtr**
  Definition of a pointer to an **HxRegValue** (p. 1104).
- typedef HxRegValueList::iterator **HxRegValueListIter**
  Iterator for **HxRegValueList** (p. 1107).
- typedef HxRegValueList::const_iterator **HxRegValueListConstIter**
  Const iterator for **HxRegValueList** (p. 1107).
- typedef **HxRegValueList::back_insert_iterator HxRegValueListBackInserter**
  Back inserter for **HxRegValueList** (p. 1107).

7.226.1 Detailed Description

7.226.2 Typedef Documentation

7.226.2.1 typedef HxRegValue* HxRegValuePtr

Definition of a pointer to an **HxRegValue** (p. 1104).

7.226.2.2 typedef HxRegValueList::iterator HxRegValueListIter

Iterator for **HxRegValueList** (p. 1107).

7.226.2.3 typedef HxRegValueList::const_iterator HxRegValueListConstIter

Const iterator for **HxRegValueList** (p. 1107).

7.226.2.4 typedef HxRegValueList::back_insert_iterator HxRegValueListBackInserter

Back inserter for **HxRegValueList** (p. 1107).

7.227 HxRestrict.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxRestrict** (**HxImageRep** img, **HxPoint** begin, **HxPoint** end)

  Restriction of domain.

7.227.1 Detailed Description

7.227.2 Function Documentation

7.227.2.1 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     HxString fname("HxRestrict");
15 
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
```

```
20      }
21      if (begin.x() < 0)
22      {
23          HxGlobalError::instance()->reportError(fname, "begin.x less then 0", HxGlobalError::HX_GE_INVALID);
24          return HxImageRep();
25      }
26      if (begin.y() < 0)
27      {
28          HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
29          return HxImageRep();
30      }
31      if ((img.dimensionality() > 2) && (begin.z() < 0))
32      {
33          HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
34          return HxImageRep();
35      }
36
37      if (end.x() < begin.x())
38      {
39          HxGlobalError::instance()->reportError(fname, "end.x less than begin.x", HxGlobalError::HX_GE_INVALID);
40          return HxImageRep();
41      }
42      if (end.y() < begin.y())
43      {
44          HxGlobalError::instance()->reportError(fname, "end.y less than begin.y", HxGlobalError::HX_GE_INVALID);
45          return HxImageRep();
46      }
47      if ((img.dimensionality() > 2) && (end.z() < begin.z()))
48      {
49          HxGlobalError::instance()->reportError(fname, "end.z less than begin.z", HxGlobalError::HX_GE_INVALID);
50          return HxImageRep();
51      }
52      if (begin.x() > img.sizes().x())
53      {
54          HxGlobalError::instance()->reportError(fname, "begin.x greater then image size", HxGlobalError::HX_GE_INVALI
55          return HxImageRep();
56      }
57      if (begin.y() > img.sizes().y())
58      {
59          HxGlobalError::instance()->reportError(fname, "begin.y greater then image size", HxGlobalError::HX_GE_INVALI
60          return HxImageRep();
61      }
62      if ((img.dimensionality() > 2) && (begin.z() > img.sizes().z()))
63      {
64          HxGlobalError::instance()->reportError(fname, "begin.z greater then image size", HxGlobalError::HX_GE_INVALI
65          return HxImageRep();
66      }
67      if (end.x() > img.sizes().x())
68      {
69          HxGlobalError::instance()->reportError(fname, "end.x greater then image size", HxGlobalError::HX_GE_INVALID)
70          return HxImageRep();
71      }
72      if (end.y() > img.sizes().y())
73      {
74          HxGlobalError::instance()->reportError(fname, "end.y greater then image size", HxGlobalError::HX_GE_INVALID)
75          return HxImageRep();
76      }
77      if ((img.dimensionality() > 2) && (end.z() > img.sizes().z()))
78      {
79          HxGlobalError::instance()->reportError(fname, "end.z greater then image size", HxGlobalError::HX_GE_INVALID)
80          return HxImageRep();
81      }
82
83      return img.restrict(begin, end);
84  }
```

7.228 HxRGB2Intensity.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxRGB2Intensity** (**HxImageRep** im)
  Conversion of RGB to intensity.

7.228.1 Detailed Description

7.228.2 Function Documentation

7.228.2.1 HxImageRep L_HXIMAGEREP HxRGB2Intensity (HxImageRep *im*)

Conversion of RGB to intensity.

The function converts an RGB image to an intensity image

```
80 {
81     HxString fname("HxRGB2Intensity");
82
83     if (im.isNull())
84     {
85         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
86         return HxImageRep();
87     }
88     if (im.signature().pixelDimensionality() != 3)
89     {
90         HxGlobalError::instance()->reportError(fname, "RGB to intensity conversions are only valid for Vec3 images",
91         return HxImageRep();
92     }
93
94     return im.unaryPixOp("RGB2Intensity");
95 }
```

7.229 HxRightShift.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxRightShift** (**HxImageRep** im1, **HxImageRep** im2)
  Right shift.

7.229.1 Detailed Description

7.229.2 Function Documentation

7.229.2.1 HxImageRep L_HXIMAGEREP HxRightShift (HxImageRep *im1*, HxImageRep *im2*)

Right shift.

The function performs addition (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoRightShift** (p. 458). The image functor instantiator : **HxInstantiatorRightShift** (p. 908).

```
13 {
14     HxString fname("HxRightShift");
15
16     if (im1.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (im2.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26
27     if (im1.signature().imageDimensionality() != im2.signature().imageDimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32     if (im1.signature().pixelDimensionality() != im2.signature().pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35         return HxImageRep();
36     }
37     if (im1.signature().pixelDimensionality() != 1)
38     {
39         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
40         return HxImageRep();
41     }
42
43     if (im1.signature().pixelType() != im2.signature().pixelType())
44     {
45         HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
46         return HxImageRep();
47     }
48     if (im1.signature().pixelType() != INT_VALUE)
49     {
50         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer values", HxGlobalE
51         return HxImageRep();
52     }
53
54     if (im1.sizes().x() != im2.sizes().x())
55     {
56         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
57         return HxImageRep();
58     }
59     if (im1.sizes().y() != im2.sizes().y())
60     {
61         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
```

```
62          return HxImageRep();
63      }
64      if (im1.signature().imageDimensionality() > 2)
65      {
66          if (im1.sizes().z() != im2.sizes().z())
67          {
68              HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
69              return HxImageRep();
70          }
71      }
72
73      return im1.binaryPixOp(im2, "rightShift");
74  }
```

7.230 HxRightShiftVal.h File Reference

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

Functions

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

  Right shift.

7.230.1 Detailed Description

7.230.2 Function Documentation

7.230.2.1 HxImageRep L_HXIMAGEREP HxRightShiftVal (HxImageRep *im*, HxValue *val*)

Right shift.

The function performs addition (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoRightShift** (p. 458). The image functor instantiator : **HxInstantiatorRightShiftV** (p. 909).

```
13  {
14      HxString fname("HxRighShiftVal");
15
16      if (im.isNull())
17      {
18          HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19          return HxImageRep();
20      }
21
22      if (im.signature().pixelDimensionality() != 1)
23      {
24          HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
25          return HxImageRep();
26      }
27
28      if (im.signature().pixelType() != INT_VALUE)
29      {
30          HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
```

```
31         return HxImageRep();
32     }
33
34     if (val.tag() != HxValue::SI)
35     {
36         HxGlobalError::instance()->reportError(fname, "only scalar integer value supported", HxGlobalError::HX_GE_IN
37         return HxImageRep();
38     }
39
40     return im.binaryPixOp(val, "rightShift");
41 }
```

7.231 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).

7.231.1 Detailed Description

7.231.2 Function Documentation

7.231.2.1 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     HxString fname("HxRotate");
18
19     if (img.isNull())
20     {
21         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
22         return HxImageRep();
23     }
24
25     if (img.dimensionality() == 2) {
26         HxMatrix m =
27             HxMatrix::translate2d(
28                 img.dimensionSize(1)/2, img.dimensionSize(2)/2) *
29             HxMatrix::rotate2dDeg(-alpha) * // Y-axis points down
30             HxMatrix::translate2d(
31                 -img.dimensionSize(1)/2, -img.dimensionSize(2)/2);
32         return img.geometricOp2d(m, gi, FORWARD, adjustSize, background);
33     } else {
34         HxEnvironment::instance()->errorStream()
35             << "3d rotation not implemented yet" << STD_ENDL;
36         HxEnvironment::instance()->flush();
37         return HxImageRep();
```

```
38     }
39 }
```

7.232 HxRound.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxRound** (**HxImageRep** im)

  Rounding.

7.232.1 Detailed Description

7.232.2 Function Documentation

7.232.2.1 HxImageRep L_HXIMAGEREP HxRound (HxImageRep *im*)

Rounding.

The function computes the rounding (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoRound** (p. 1241). The image functor instantiator : **HxInstantiatorRound** (p. 910).

```
13 {
14     HxString fname("HxRound");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("round");
23 }
```

7.233 HxScale.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxScale** (**HxImageRep** img, double sx, double sy, double sz=1.0, **HxGeoIntType** gi=LINEAR, int adjustSize=1)

  Scaling.

7.233.1 Detailed Description

7.233.2 Function Documentation

7.233.2.1 HxImageRep L_HXIMAGEREP HxScale (HxImageRep *img*, double *sx*, double *sy*, double *sz* = 1.0, HxGeoIntType *gi* = LINEAR, int *adjustSize* = 1)

Scaling.

```
16 {
17     HxString fname("HxScale");
18
19     if (img.isNull())
20     {
21         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
22         return HxImageRep();
23     }
24
25     if (img.dimensionality() == 2) {
26         HxMatrix m = HxMatrix::scale2d(sx, sy);
27         return img.geometricOp2d(m, gi, FORWARD, adjustSize);
28     } else {
29         HxEnvironment::instance()->errorStream()
30             << "3d scaling not implemented yet" << STD_ENDL;
31         HxEnvironment::instance()->flush();
32         return HxImageRep();
33     }
34 }
```

7.234 HxSegmentationCentralMoments.h File Reference

Functions

- void **HxSegmentationCentralMoments** (**HxSegmentation2d** ∗seg, int order)

  Compute the central moments of each blob in an **HxSegmentation2d** (p. 1182).

7.234.1 Detailed Description

7.234.2 Function Documentation

7.234.2.1 void HxSegmentationCentralMoments (HxSegmentation2d ∗ *seg*, int *order*)

Compute the central moments of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature ”CentralMoments” in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValueList vl = HxIdentMaskCentralMoments(seg->getInputImage(),
21                                                    seg->getLabeledImage(),
22                                                    blob->startMaer(),
23                                                    blob->sizeMaer(),
24                                                    blob->getLabel(),
```

```
25                                                  order);
26         blob->addFeature("CentralMoments", vl);
27     }
28 }
```

7.235 HxSegmentationHistogram.h File Reference

Functions

- void **HxSegmentationHistogram** (**HxSegmentation2d** ∗seg, int getDim, double lowBin, double highBin, int nBin)

  Compute the histogram of each blob in an **HxSegmentation2d** (p. 1182).

7.235.1 Detailed Description

7.235.2 Function Documentation

7.235.2.1 void HxSegmentationHistogram (HxSegmentation2d ∗ *seg*, int *getDim*, double *lowBin*, double *highBin*, int *nBin*)

Compute the histogram of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature ”Histogram” in a blob.

```
20 {
21     HxImageRep inputIm = seg->getInputImage();
22     HxImageRep labeledIm = seg->getLabeledImage();
23     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
24         HxBlob2d* blob = (*b);
25         int nDim = (getDim <= 0) ? inputIm.pixelDimensionality() : 1;
26         HxHistogram hist = HxHistogram(REAL_VALUE, nDim, lowBin, highBin, nBin,
27                                        lowBin, highBin, nBin,
28                                        lowBin, highBin, nBin);
29         HxBoundingBox bb(blob->sizeMaer());
30         bb = bb.translate(blob->startMaer());
31
32         HxTagList tags;
33         HxAddTag(tags, "histogram", &hist);
34         HxAddTag(tags, "getDim", getDim);
35         HxAddTag(tags, "maskVal", blob->getLabel());
36         HxAddTag(tags, "boundingBox", bb);
37         inputIm.exportOpExtra("histogramMask", labeledIm, tags);
38         blob->addFeature("Histogram", hist);
39     }
40 }
```

7.236 HxSegmentationMean.h File Reference

Functions

- void **HxSegmentationMean** (**HxSegmentation2d** ∗seg)

  Compute the mean of each blob in an **HxSegmentation2d** (p. 1182).

7.236.1 Detailed Description

7.236.2 Function Documentation

7.236.2.1 void HxSegmentationMean (HxSegmentation2d * *seg*)

Compute the mean of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature ”Mean” in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValue v = HxIdentMaskMean(seg->getInputImage(), seg->getLabeledImage(),
21                                     blob->startMaer(), blob->sizeMaer(),
22                                     blob->getLabel());
23         blob->addFeature("Mean", v);
24     }
25 }
```

7.237 HxSegmentationMedian.h File Reference

Functions

- void **HxSegmentationMedian** (**HxSegmentation2d** *seg)

  Compute the median of each blob in an **HxSegmentation2d** (p. 1182).

7.237.1 Detailed Description

7.237.2 Function Documentation

7.237.2.1 void HxSegmentationMedian (HxSegmentation2d * *seg*)

Compute the median of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature ”Median” in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValue v = HxIdentMaskMedian(seg->getInputImage(), seg->getLabeledImage(),
21                                       blob->startMaer(), blob->sizeMaer(),
22                                       blob->getLabel());
23         blob->addFeature("Median", v);
24     }
25 }
```

7.238 HxSegmentationMoments.h File Reference

Functions

- void **HxSegmentationMoments** (**HxSegmentation2d** *seg, int order)

Compute the moments of each blob in an **HxSegmentation2d** (p. 1182).

7.238.1 Detailed Description

7.238.2 Function Documentation

7.238.2.1 void HxSegmentationMoments (HxSegmentation2d * *seg*, int *order*)

Compute the moments of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature "Moments" in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValueList vl = HxIdentMaskMoments(seg->getInputImage(),
21                                             seg->getLabeledImage(),
22                                             blob->startMaer(), blob->sizeMaer(),
23                                             blob->getLabel(), order);
24         blob->addFeature("Moments", vl);
25     }
26 }
```

7.239 HxSegmentationStDev.h File Reference

Functions

- void **HxSegmentationStDev** (**HxSegmentation2d** *seg)

  Compute the standard deviation of each blob in an **HxSegmentation2d** (p. 1182).

7.239.1 Detailed Description

7.239.2 Function Documentation

7.239.2.1 void HxSegmentationStDev (HxSegmentation2d * *seg*)

Compute the standard deviation of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature "StDev" in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValue v = HxIdentMaskStDev(seg->getInputImage(), seg->getLabeledImage(),
21                                      blob->startMaer(), blob->sizeMaer(),
22                                      blob->getLabel());
23         blob->addFeature("StDev", v);
24     }
25 }
```

7.240 HxSegmentationSum.h File Reference

Functions

- void **HxSegmentationSum** (**HxSegmentation2d** ∗seg)

  Compute the sum of each blob in an **HxSegmentation2d** (p. 1182).

7.240.1 Detailed Description

7.240.2 Function Documentation

7.240.2.1 void HxSegmentationSum (HxSegmentation2d ∗ *seg*)

Compute the sum of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature ”Sum” in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValue v = HxIdentMaskSum(seg->getInputImage(), seg->getLabeledImage(),
21                                    blob->startMaer(), blob->sizeMaer(),
22                                    blob->getLabel());
23         blob->addFeature("Sum", v);
24     }
25 }
```

7.241 HxSegmentationVariance.h File Reference

Functions

- void **HxSegmentationVariance** (**HxSegmentation2d** ∗seg)

  Compute the variance of each blob in an **HxSegmentation2d** (p. 1182).

7.241.1 Detailed Description

7.241.2 Function Documentation

7.241.2.1 void HxSegmentationVariance (HxSegmentation2d ∗ *seg*)

Compute the variance of each blob in an **HxSegmentation2d** (p. 1182).

The result is stored as feature ”Variance” in a blob.

```
17 {
18     for (HxBlob2dListConstIter b=seg->getBlobBegin() ; b<seg->getBlobEnd() ; b++) {
19         HxBlob2d* blob = (*b);
20         HxValue v = HxIdentMaskVariance(seg->getInputImage(), seg->getLabeledImage(),
21                                         blob->startMaer(), blob->sizeMaer(),
22                                         blob->getLabel());
```

```
23         blob->addFeature("Variance", v);
24     }
25 }
```

7.242 HxSetBorderValue.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxSetBorderValue** (**HxImageRep** im, int w, int h, **HxValue** val)

  SetBorderValue.

7.242.1 Detailed Description

7.242.2 Function Documentation

7.242.2.1 HxImageRep L_HXIMAGEREP HxSetBorderValue (HxImageRep *im*, int *w*, int *h*, HxValue *val*)

SetBorderValue.

The function sets all pixels at the border to the given value.

```
17 {
18     int x1,y1,x2,y2;
19     HxSizes sz=im.sizes();
20     x1 = w;
21     y1 = h;
22     x2 = sz.x()-w;
23     y2 = sz.y()-h;
24
25     HxTagList tags;
26     HxAddTag(tags, "x1", x1);
27     HxAddTag(tags, "y1", y1);
28     HxAddTag(tags, "x2", x2);
29     HxAddTag(tags, "y2", y2);
30     HxAddTag(tags, "val", val);
31
32     HxAddTag(tags, "inside", false);
33
34     return im.unaryPixOp("setPartImg", tags);
35 }
```

7.243 HxSetPartImage.c File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxSetPartImage** (**HxImageRep** im, int x1, int y1, int x2, int y2, **HxValue** val)

  SetPartImage.

7.243.1 Detailed Description

7.243.2 Function Documentation

7.243.2.1 HxImageRep L_HXIMAGEREP HxSetPartImage (HxImageRep *im*, int *x1*, int *y1*, int *x2*, int *y2*, HxValue *val*)

SetPartImage.

The function replaces all pixels in rectange (x1,y1,x2,y2) with the given value "val".

```
25 {
26     HxTagList tags;
27     HxAddTag(tags, "x1", x1);
28     HxAddTag(tags, "y1", y1);
29     HxAddTag(tags, "x2", x2);
30     HxAddTag(tags, "y2", y2);
31     HxAddTag(tags, "val", val);
32
33     HxAddTag(tags, "inside", true);
34
35     return im.unaryPixOp("setPartImg", tags);
36 }
```

7.244 HxSetPartImage.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxSetPartImage** (**HxImageRep** im, int x1, int y1, int x2, int y2, **HxValue** val)

  SetPartImage.

7.244.1 Detailed Description

7.244.2 Function Documentation

7.244.2.1 HxImageRep L_HXIMAGEREP HxSetPartImage (HxImageRep *im*, int *x1*, int *y1*, int *x2*, int *y2*, HxValue *val*)

SetPartImage.

The function replaces all pixels in rectange (x1,y1,x2,y2) with the given value "val".

7.245 HxSin.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxSin** (**HxImageRep** im)

  Sine.

7.245.1 Detailed Description

7.245.2 Function Documentation

7.245.2.1 HxImageRep L_HXIMAGEREP HxSin (HxImageRep *im*)

Sine.

The function computes the sine (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoSin** (p. 1242). The image functor instantiator : **HxInstantiatorSin** (p. 915).

```
13 {
14     HxString fname("HxSin");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("sin");
23 }
```

7.246 HxSinh.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxSinh** (**HxImageRep** im)

  Hyperbolic sine.

7.246.1 Detailed Description

7.246.2 Function Documentation

7.246.2.1 HxImageRep L_HXIMAGEREP HxSinh (HxImageRep *im*)

Hyperbolic sine.

The function computes the hyperbolic sine (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoSinh** (p. 1244). The image functor instantiator : **HxInstantiatorSinh** (p. 916).

```
13 {
14     HxString fname("HxSinh");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("sinh");
23 }
```

7.247 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** &)

7.247.1 Detailed Description

7.247.2 Typedef Documentation

7.247.2.1 typedef HxVec3Int HxSizes

Definition of sizes.

HxSizes is used primarily to specify the dimension sizes of an image or a region in an image.

7.248 HxSkeleton.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSkeleton** (**HxImageRep** im, **HxSF** sf)

  This should be renamed as HxMedialAxisTransform and we should implement HxSkeleton correctly be aware of changes in object topology.

7.248.1 Detailed Description

7.248.2 Function Documentation

7.248.2.1 HxImageRep L_HXIMAGEREP HxSkeleton (HxImageRep *im*, HxSF *sf*)

This should be renamed as HxMedialAxisTransform and we should implement HxSkeleton correctly be aware of changes in object topology.

function y=mmskelm_equ(f, B) y = mmbinary(zeros(size(f))); for i=0:length(f) nb = mmsesum(B,i); f1 = mmero(f,nb); f2 = mmopenth(f1,B); y = mmunion(y,f2); end;

```
34 {
35     HxImageRep res, tmp;
36 
37     res= HxMakeFromValue(im.signature(), im.sizes(), 0);
38 
39     HxSF sfn;
40 
41 //  for(int i=0; i<im.numberOfPixels(); i++)
42     for(int i=0; i<15; i++)
43     {
44         sfn = sf.dilateSF(i);
45         tmp = HxOpeningTopHat(HxErosion(im, sfn) , sf);
46         res = HxMax(tmp, res);
47     }
48 
49     return res;
50 }
```

7.249 HxSKIZ.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSKIZ** (**HxImageRep** input, int conn)

  using the Luc Vincent watershed.

7.249.1 Detailed Description

7.249.2 Function Documentation

7.249.2.1 HxImageRep L_HXIMAGEREP HxSKIZ (HxImageRep *input*, int *conn*)

using the Luc Vincent watershed.

```
105 {
106     HxImageRep mask=input;
107     HxTagList tags;
108     HxAddTag(tags, "connectivity", conn);
109 
110     //to speedup compute here the min and max gray values
111     int     hmin,hmax;
112     hmin = HxPixMin(input).HxScalarIntValue().x();
113     hmax = HxPixMax(input).HxScalarIntValue().x();
114     HxAddTag(tags,"hmin",hmin);
115     HxAddTag(tags,"hmax",hmax);
116 
117     //for more speedup, use a histogram to indicate the existing gray levels
118 
119     HxImageRep out = input.queueBasedOp(mask, "qWaterShedLV", tags);
120 
121     //because the LucVincent algorithm doesn't get all the watershed points
122     //this neigborhoud operator replaces the smallest label with wshed value
123     //out =  HxLWshed(out,conn,WSHEDVAL);
124     HxAddTag(tags, "conn", conn);
125     HxAddTag(tags, "wshedval", WSHEDVAL);
126 
127     out = out.neighbourhoodOp("lwshed", tags);
128 
129 //now for SKIZ I have to replace all values BUT 0 with 255, so that we
130 //keep only the border, and not the objects labeled.
131 //this we can do with a binary pixel opeartion implementing a LUT (look-up table) operation
132 
133     HxTagList tags2;
134     HxAddTag(tags2, "holdValue", 0);
135     HxAddTag(tags2, "newValue", 255);
136 
137     return out.unaryPixOp("ReplaceAllValuesButOne", tags2);
138 
139 }
```

7.250 HxSqrt.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxSqrt** (**HxImageRep** im)

  Square root.

7.250.1 Detailed Description

7.250.2 Function Documentation

7.250.2.1 HxImageRep L_HXIMAGEREP HxSqrt (HxImageRep *im*)

Square root.

The function computes the square root (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoSqrt** (p. 1245). The image functor instantiator : **HxInstantiatorSqrt** (p. 918).

```
13 {
14     HxString fname("HxSqrt");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     HxValue vinf = HxPixInf(im);
22
23     if (im.signature().pixelDimensionality() == 1)
24     {
25         if (((HxScalarDouble) vinf) <= 0.0)
26         {
27             HxGlobalError::instance()->reportError(fname, im.name(), "values less than 0", HxGlobalError::HX_GE_OUTO
28         }
29     }
30     else if (im.signature().pixelDimensionality() == 2)
31     {
32         HxVec2Double vinf2d = (HxVec2Double) vinf;
33         if ((vinf2d.x() <= 0.0) || (vinf2d.y() <= 0.0))
34         {
35             HxGlobalError::instance()->reportError(fname, im.name(), "2D values are less than 0", HxGlobalError::HX_
36         }
37     }
38     else if (im.signature().pixelDimensionality() == 3)
39     {
40         HxVec3Double vinf3d = (HxVec3Double) vinf;
41         if ((vinf3d.x() <= 0.0) || (vinf3d.y() <= 0.0) || (vinf3d.z() <= 0.0))
42         {
43             HxGlobalError::instance()->reportError(fname, im.name(), "3D values are less than 0", HxGlobalError::HX_
44         }
45     }
46
47     return im.unaryPixOp("sqrt");
48 }
```

7.251 HxSquaredDistance.h File Reference

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

Functions

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

Squared distance.

7.251.1 Detailed Description

7.251.2 Function Documentation

7.251.2.1 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. 459). The image functor instantiator : **HxInstantiatorSqrDst** (p. 917).

```
103 {
104     HxString fname("HxSquaredDistance");
105 
106     if (im1.isNull())
107     {
108         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
109         return HxImageRep();
110     }
111     if (im2.isNull())
112     {
113         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
114         return HxImageRep();
115     }
116 
117     if (im1.dimensionality() != im2.dimensionality())
118     {
119         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUA
120         return HxImageRep();
121     }
122     if (im1.pixelDimensionality() != im2.pixelDimensionality())
123     {
124         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionality", HxGlobalError::HX_GE_UNEQUAL_
125         return HxImageRep();
126     }
127 
128     if (im1.sizes().x() != im2.sizes().x())
129     {
130         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
131         return HxImageRep();
132     }
133 
134     if (im1.sizes().y() != im2.sizes().y())
135     {
136         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES)
137         return HxImageRep();
138     }
139     if (im1.dimensionality() > 2)
140     {
141         if (im1.sizes().z() != im2.sizes().z())
142         {
143             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAG
144             return HxImageRep();
145         }
```

```
146     }
147
148     // call HxImageRep member function to do the image processing
149     return im1.binaryPixOp(im2, "sqrDst");
150 }
```

7.252 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. 1191).

- typedef HxStringList::const_iterator **HxStringListConstIter**

  Const iterator for **HxStringList** (p. 1191).

- typedef **HxStringList::back_insert_iterator HxStringListBackInserter**

  Back inserter for **HxStringList** (p. 1191).

Functions

- **HxString makeString** (**HxStringListConstIter** begin, **HxStringListConstIter** end, **HxString** separator=**HxString**(""))
- int **splitString** (**HxString** src, char separator, **HxStringListBackInserter**)

7.252.1 Detailed Description

7.252.2 Typedef Documentation

7.252.2.1 typedef HxStringList::iterator HxStringListIter

Iterator for **HxStringList** (p. 1191).

7.252.2.2 typedef HxStringList::const_iterator HxStringListConstIter

Const iterator for **HxStringList** (p. 1191).

7.252.2.3 typedef HxStringList::back_insert_iterator HxStringListBackInserter

Back inserter for **HxStringList** (p. 1191).

7.253 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)

7.253.1 Detailed Description

7.253.2 Typedef Documentation

7.253.2.1 typedef std::string HxString

HxString definition.

7.254 HxSub.h File Reference

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

Functions

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

  Subtraction.

7.254.1 Detailed Description

7.254.2 Function Documentation

7.254.2.1 HxImageRep L_HXIMAGEREP HxSub (HxImageRep *im1*, HxImageRep *im2*)

Subtraction.

The function performs subtraction (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoSub** (p. 461). The image functor instantiator : **HxInstantiatorSub** (p. 918).

```
16 {
17     HxString fname("HxSub");
18
19     if (im1.isNull())
20     {
21         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
22         return HxImageRep();
23     }
24     if (im2.isNull())
25     {
26         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
27         return HxImageRep();
28     }
29
30     if (im1.signature().imageDimensionality() != im2.signature().imageDimensionality())
31     {
32         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
33         return HxImageRep();
34     }
35     if (im1.signature().pixelDimensionality() != im2.signature().pixelDimensionality())
36     {
37         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
38         return HxImageRep();
39     }
40 //  if (im1.signature().pixelType() != im2.signature().pixelType())
41 //  {
42 //      HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGETYPES
43 //      return HxImageRep();
44 //  }
45 //  if (im1.signature().pixelPrecision() != im2.signature().pixelPrecision())
46 //  {
47 //      HxGlobalError::instance()->reportError(fname, "unequal pixel precisions", HxGlobalError::HX_GE_UNEQUAL_IMAGE
48 //      return HxImageRep();
49 //  }
50
51     if (im1.sizes().x() != im2.sizes().x())
52     {
53         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
54         return HxImageRep();
55     }
56     if (im1.sizes().y() != im2.sizes().y())
57     {
58         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
59         return HxImageRep();
60     }
61     if (im1.signature().imageDimensionality() > 2)
62     {
63         if (im1.sizes().z() != im2.sizes().z())
64         {
```

```
65              HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
66              return HxImageRep();
67          }
68      }
69
70      // in case of byte, unsigned: generate warnings in case of potentially dangerous
71      // situations.
72      // Check if image is byte.
73      if (((im1.signature().pixelType() == INT_VALUE) &&
74           (im1.signature().pixelPrecision() == 8)) ||
75          ((im2.signature().pixelType() == INT_VALUE) &&
76           (im2.signature().pixelPrecision() == 8)))
77      {
78          if ((im1.pixelDimensionality() == 1) && (im1.pixelDimensionality() == 1))
79          {
80              if ((HxPixMin(im1).HxScalarIntValue() -
81                   HxPixMin(im2).HxScalarIntValue()) < HxScalarInt(0))
82              {
83                  HxGlobalError::instance()->reportWarning(fname,
84                      im1.name()+HxString(" ")+im2.name(),
85                      "possible underflow due to byte precision",
86                      HxGlobalError::HX_GW_OVERFLOW);
87              }
88          }
89          else if ((HxPixMin(HxUnaryMin(im1)).HxScalarIntValue() -
90                    HxPixMin(HxUnaryMin(im2)).HxScalarIntValue()) < HxScalarInt(0))
91          {
92                  HxGlobalError::instance()->reportWarning(fname,
93                      im1.name()+HxString(" ")+im2.name(),
94                      "possible underflow due to byte precision",
95                      HxGlobalError::HX_GW_OVERFLOW);
96          }
97      }
98
99
100     return im1.binaryPixOp(im2, "sub");
101 }
```

7.255 HxSubSat.h File Reference

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

Functions

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

  Saturated subtraction.

7.255.1 Detailed Description

7.255.2 Function Documentation

7.255.2.1 HxImageRep L_HXIMAGEREP HxSubSat (HxImageRep *im1*, HxImageRep *im2*)

Saturated subtraction.

The function computes the saturated subtraction of all corresponding pixels in the input images via a binary pixel operation.

```
130 {
131     // call HxImageRep member function to do the image processing
132 //  std::cout << im1.signature() << std::endl;
133 //  std::cout << im2.signature() << std::endl;
134
135     HxTagList tl;
136     HxAddTag(tl,"minSat", 0);//HxPixMin(im1).HxScalarIntValue().x() );
137
138
139     return im1.binaryPixOp(im2, "subSat", tl);
140 }
```

7.256 HxSubVal.h File Reference

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

Functions

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

  Subtraction.

7.256.1 Detailed Description

7.256.2 Function Documentation

7.256.2.1 HxImageRep L_HXIMAGEREP HxSubVal (HxImageRep *im*, HxValue *val*)

Subtraction.

The function performs subtraction (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoSub** (p. 461). The image functor instantiator : **HxInstantiatorSubV** (p. 920).

```
13 {
14     HxString fname("HxSubVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
```

```
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
35     if (im.signature().pixelDimensionality() != valdim)
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "sub");
43 }
```

7.257 HxSup.h File Reference

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

Functions

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

  Supremum.

7.257.1 Detailed Description

7.257.2 Function Documentation

7.257.2.1 HxImageRep L_HXIMAGEREP HxSup (HxImageRep *im1*, HxImageRep *im2*)

Supremum.

The function performs supremum (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoSup** (p. 466). The image functor instantiator : **HxInstantiatorSup** (p. 921).

```
14 {
15     HxString fname("HxSup");
16
17     if (im1.isNull())
18     {
19         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
20         return HxImageRep();
21     }
22     if (im2.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return HxImageRep();
26     }
27
28     if (im1.dimensionality() != im2.dimensionality())
29     {
```

```
30          HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
31          return HxImageRep();
32      }
33      if (im1.pixelDimensionality() != im2.pixelDimensionality())
34      {
35          HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
36          return HxImageRep();
37      }
38
39      if (im1.sizes().x() != im2.sizes().x())
40      {
41          HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42          return HxImageRep();
43      }
44      if (im1.sizes().y() != im2.sizes().y())
45      {
46          HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47          return HxImageRep();
48      }
49      if (im1.dimensionality() > 2)
50      {
51          if (im1.sizes().z() != im2.sizes().z())
52          {
53              HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGE
54              return HxImageRep();
55          }
56      }
57
58      return im1.binaryPixOp(im2, "sup");
59  }
```

7.258 HxSupremumReconstruction.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxSupremumReconstruction** (**HxImageRep** im, **HxImageRep** mask, **HxSF** sf)

  function y=mmsuprec_equ(f, g, bc) n = length(f); y = mmcero(f,g,bc,n);.

7.258.1 Detailed Description

7.258.2 Function Documentation

7.258.2.1 HxImageRep L_HXIMAGEREP HxSupremumReconstruction (HxImageRep *im*, HxImageRep *mask*, HxSF *sf*)

function y=mmsuprec_equ(f, g, bc) n = length(f); y = mmcero(f,g,bc,n);.

```
25 {
26      HxImageRep res;
27
28      //this might be a huge number.
29      //defintion is to do conditional dilation untill stability
```

```
30     //I should check for stability here, or in ConditionalDilation
31     //by comparing two subsequent images
32     //Because this is also expesinve, I might check only after every 10 iterations
33
34     int n = im.dimensionSize(1)*im.dimensionSize(2);
35
36     res = HxConditionalErosion(im, mask, sf, n);
37     return res;
38 }
```

7.259 HxSupVal.h File Reference

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

Functions

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

  Supremum.

7.259.1 Detailed Description

7.259.2 Function Documentation

7.259.2.1 HxImageRep L_HXIMAGEREP HxSupVal (HxImageRep *im*, HxValue *val*)

Supremum.

The function performs supremum (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoSup** (p. 466). The image functor instantiator : **HxInstantiatorSupV** (p. 922).

```
13 {
14     HxString fname("HxSupVal");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     int valdim;
23     if ((val.tag() == HxValue::SI) || (val.tag() == HxValue::SD))
24     {
25         valdim = 1;
26     }
27     else if ((val.tag() == HxValue::V2I) || (val.tag() == HxValue::V2D))
28     {
29         valdim = 2;
30     }
31     else
32     {
33         valdim = 3;
34     }
```

```
35     if (im.signature().pixelDimensionality() != valdim)
36     {
37         HxGlobalError::instance()->reportError(fname, "pixel dimensionality differs from value dimensionality",
38                         HxGlobalError::HX_GE_UNEQUAL_DIMS);
39         return HxImageRep();
40     }
41
42     return im.binaryPixOp(val, "sup");
43 }
```

7.260 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.

7.260.1 Detailed Description

7.260.2 Function Documentation

7.260.2.1 template<class ValT> void HxAddTag (HxTagList & *tags*, HxString *name*, ValT *v*) [inline]

Add a tag to the list of tags.

```
92 {
93     typedef HxTagTem<ValT>  Tag;
94     tags.addTag(new Tag(name, v));
95 }
```

7.260.2.2 template<class ValT> ValT HxGetTag (const HxTagList & *tags*, HxString *name*) [inline]

Get a tag from the list of tags.

```
102 {
103     typedef HxTagTem<ValT>* TagPtr;
104     TagPtr t = TagPtr(tags.getTag(name));
105     ValT v;
106     if (t)
107         v = t->getValue();
108     return v;
109 }
```

7.260.2.3 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.

```
117 {
118     typedef HxTagTem<ValT>* TagPtr;
119     TagPtr t = TagPtr(tags.getTag(name));
120     return t ? t->getValue() : defV;
121 }
```

7.260.2.4 bool HxTagIsSet (const HxTagList & *tags*, HxString *name*) [inline]

Check if tag is set.

```
127 {
128     return tags.getTag(name) ? true : false;
129 }
```

7.260.2.5 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 }
```

7.261 HxTan.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxTan** (**HxImageRep** im)

  Tangent.

7.261.1 Detailed Description

7.261.2 Function Documentation

7.261.2.1 HxImageRep L_HXIMAGEREP HxTan (HxImageRep *im*)

Tangent.

The function computes the tangent (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoTan** (p. 1248). The image functor instantiator : **HxInstantiatorTan** (p. 923).

```
13 {
14     HxString fname("HxTan");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("tan");
23 }
```

7.262 HxTanh.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxTanh** (**HxImageRep** im)

  Hyperbolic tangent.

7.262.1 Detailed Description

7.262.2 Function Documentation

7.262.2.1 HxImageRep L_HXIMAGEREP HxTanh (HxImageRep *im*)

Hyperbolic tangent.

The function computes the hyperbolic tangent (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoTanh** (p. 1250). The image functor instantiator : **HxInstantiatorTanh** (p. 923).

```
13 {
14     HxString fname("HxTanh");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21
22     return im.unaryPixOp("tanh");
23 }
```

7.263 HxThickening.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxThickening** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)

  This should be implemented (Rein) as repetitive substraction from the image of result of HitOrMiss.

7.263.1 Detailed Description

7.263.2 Function Documentation

7.263.2.1 HxImageRep L_HXIMAGEREP HxThickening (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

This should be implemented (Rein) as repetitive substraction from the image of result of HitOrMiss.

As they are not rotation invariant, the structuring elements of have to be rotated

```
13 {
14     return im;
15 }
```

7.264 HxThinning.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxThinning** (**HxImageRep** im, **HxSF** sf1, **HxSF** sf2)
  This should be implemented (Rein) as repetitive substraction from the image of result of HitOrMiss.

7.264.1 Detailed Description

7.264.2 Function Documentation

7.264.2.1 HxImageRep L_HXIMAGEREP HxThinning (HxImageRep *im*, HxSF *sf1*, HxSF *sf2*)

This should be implemented (Rein) as repetitive substraction from the image of result of HitOrMiss.

As they are not rotation invariant, the structuring elements of have to be rotated

```
210 {
211 //  return HxSub(im,HxHitOrMiss(im,sf1,sf2));
212
213
214     HxImageSignature sig = im.signature();
215     HxSizes sizes(3, 3, 0);
216     HxValue val(0);
217 //    HxValue val(2);
218     HxSFFactory SFFi = HxSFFactory::instance();
219     HxSF sf;
220     //Prepare the flat SF
221     sf = SFFi.makeFlatSF(sig, sizes, val);
222
223
224     HxImageRep contur = HxMorphologicalContour(im,sf);
225
226     HxImageRep res = im;
227     for(int i=0;i<20;i++)
228     {
229         res = HxErosion(res,sf);
230         //res= HxSub(im,contur);
231         //contur = HxMorphologicalContour(res,sf);
232
233     }
234     return res;
235 /*
236     HxTagList tags;
237     HxAddTag(tags, "connectivity", 8);
238
239     return im.queueBasedOp(contur, "thinning", tags);
240
241 */
242 }
```

7.265 HxThreshold.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxThreshold** (**HxImageRep** im, **HxValue** level)
  Thresholding.

7.265.1 Detailed Description

7.265.2 Function Documentation

7.265.2.1 HxImageRep L_HXIMAGEREP HxThreshold (HxImageRep *im*, HxValue *level*)

Thresholding.

```
13 {
14     HxString fname("HxThreshold");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21 
22     if (im.pixelDimensionality() == 1)
23     {
24         if ((level.tag() != HxValue::SI) && (level.tag() != HxValue::SD))
25         {
26             HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlobalError::H
27         }
28     }
29     if (im.pixelDimensionality() == 2)
30     {
31         if ((level.tag() != HxValue::V2I) && (level.tag() != HxValue::V2D)
32                                           && (level.tag() != HxValue::CPL))
33         {
34             HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlobalError::H
35         }
36     }
37     if (im.pixelDimensionality() == 3)
38     {
39         if ((level.tag() != HxValue::V3I) && (level.tag() != HxValue::V3D))
40         {
41             HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlobalError::H
42         }
43     }
44 
45     HxTagList tags;
46     HxAddTag(tags, "level", level);
47     return im.unaryPixOp("threshold", tags);
48 }
```

7.266 HxTranslate.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxTranslate** (**HxImageRep** img, int doX, int doY, int doZ=1)
  Translation.

7.266.1 Detailed Description

7.266.2 Function Documentation

7.266.2.1 HxImageRep L_HXIMAGEREP HxTranslate (HxImageRep *img*, int *doX*, int *doY*, int *doZ* = 1)

Translation.

```
14 {
15     if (img.dimensionality() == 2) {
16         HxMatrix m(img.sizes().x(),img.sizes().y());
17             m = m.translate2d(doX, doY);
18         return img.geometricOp2d(m, NEAREST, FORWARD, 0, HxValue(0));
19     } else {
20         HxEnvironment::instance()->errorStream()
21             << "3d Translation not implemented yet" << STD_ENDL;
22         HxEnvironment::instance()->flush();
23         return HxImageRep();
24     }
25 }
```

7.267 HxTranspose.c File Reference

```
#include "HxTranspose.h"
#include "HxTagList.h"
#include "HxClassName.h"
#include "HxCategories.h"
#include "HxImgFtorDiy.h"
#include "HxIncludedSigs.h"
```

Namespaces

- namespace **HxInstDiyTranspose_c**

Compounds

- class **HxDiyTranspose**

Functor for transpose.

- class **HxInstDiyTranspose**

  Instantiator for DIY operation with transpose.

Functions

- template<class DstDataPtrType, class SrcDataPtrType> void **HxTranspose_Line** (DstDataPtrType dstPtr, SrcDataPtrType srcPtr, int nPix)

  Transpose one row of pixels.

- **HxImageRep HxTranspose** (**HxImageRep** im)

  Transpose.

7.267.1 Detailed Description

7.267.2 Function Documentation

7.267.2.1 template<class DstDataPtrType, class SrcDataPtrType> void HxTranspose_Line (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, int *nPix*)

Transpose one row of pixels.

```
44 {
45     while (--nPix >= 0) {
46         dstPtr.write(srcPtr.readIncX());
47         dstPtr.incY();
48     }
49 }
```

7.267.2.2 HxImageRep HxTranspose (HxImageRep *im*)

Transpose.

Implementation specifics :

- Pattern : **Do It Yourself operation** (p. 60)
- The pixel functor : **HxDiyTranspose** (p. 529)
- Instantiations : **HxInstDiyTranspose_c** (p. **??**)

```
111 {
112     HxSizes srcSizes = im.sizes();
113     HxSizes resultSizes = srcSizes;
114
115     switch (im.dimensionality())
116     {
117     case 1  :
118         break;
119     case 2  :
120         resultSizes = HxSizes(srcSizes.y(), srcSizes.x(), 1);
```

```
121         break;
122     case 3  :
123         resultSizes = HxSizes(srcSizes.y(), srcSizes.z(), srcSizes.x());
124         break;
125     }
126
127     HxTagList tags;
128     HxAddTag(tags, "resultSizes", resultSizes);
129
130     return im.diyOp("transpose", tags);
131 }
```

7.268 HxTranspose.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxTranspose** (**HxImageRep** im)

  Transpose.

7.268.1 Detailed Description

7.268.2 Function Documentation

7.268.2.1 HxImageRep L_HXIMAGEREP HxTranspose (HxImageRep *im*)

Transpose.

Implementation specifics :

- Pattern : **Do It Yourself operation** (p. 60)
- The pixel functor : **HxDiyTranspose** (p. 529)
- Instantiations : **HxInstDiyTranspose_c** (p. **??**)

7.269 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.

7.269.1 Detailed Description

7.269.2 Function Documentation

7.269.2.1 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. 1251). The image functor instantiator : **HxInstantiatorTriStateThreshold** (p. 924).

```
131 {
132     HxString fname("HxTriStateThreshold");
133 
134     if (im.isNull())
135     {
136         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
137         return HxImageRep();
138     }
139 
140     if (im.pixelDimensionality() == 1)
141     {
142         if ((level.tag() != HxValue::SI) && (level.tag() != HxValue::SD))
143         {
144             HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlobalError:
145         }
146     }
147     if (im.pixelDimensionality() == 2)
148     {
149         if ((level.tag() != HxValue::V2I) && (level.tag() != HxValue::V2D)
150                                           && (level.tag() != HxValue::CPL))
151         {
152             HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlobalError:
153         }
154     }
155     if (im.pixelDimensionality() == 3)
156     {
157         if ((level.tag() != HxValue::V3I) && (level.tag() != HxValue::V3D))
158         {
159             HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlobalError:
160         }
161     }
162 
163     // Put all non-image parameters in a TagList
164     HxTagList tags;
165     HxAddTag(tags, "level", level);
166     HxAddTag(tags, "v1", v1);
167     HxAddTag(tags, "v2", v2);
168     HxAddTag(tags, "v3", v3);
169 
170     // call HxImageRep member function to do the image processing
171     return im.unaryPixOp("triStateThreshold", tags);
172 }
```

7.270 HxUnaryMax.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnaryMax** (**HxImageRep** im)

  Unary maximum.

7.270.1 Detailed Description

7.270.2 Function Documentation

7.270.2.1 HxImageRep L_HXIMAGEREP HxUnaryMax (HxImageRep *im*)

Unary maximum.

The function computes the unary maximum (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoMax** (p. 1229). The image functor instantiator : **HxInstantiatorUpoMax** (p. 924).

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

7.271 HxUnaryMin.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnaryMin** (**HxImageRep** im)

  Unary minimum.

7.271.1 Detailed Description

7.271.2 Function Documentation

7.271.2.1 HxImageRep L_HXIMAGEREP HxUnaryMin (HxImageRep *im*)

Unary minimum.

The function computes the unary minimum (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoMin** (p. 1230). The image functor instantiator : **HxInstantiatorUpoMin** (p. 925).

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

7.272 HxUnaryProduct.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnaryProduct** (**HxImageRep** im)
  Unary product.

7.272.1 Detailed Description

7.272.2 Function Documentation

7.272.2.1 HxImageRep L_HXIMAGEREP HxUnaryProduct (HxImageRep *im*)

Unary product.

The function computes the unary product (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoProduct** (p. 1239). The image functor instantiator : **HxInstantiatorProduct** (p. 907).

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

7.273 HxUnarySum.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxUnarySum** (**HxImageRep** im)
  Unary sum.

7.273.1 Detailed Description

7.273.2 Function Documentation

7.273.2.1 HxImageRep L_HXIMAGEREP HxUnarySum (HxImageRep *im*)

Unary sum.

The function computes the unary sum (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxUpoSum** (p. 1247). The image functor instantiator : **HxInstantiatorSum** (p. 920).

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

7.274 HxUniform.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxUniform** (**HxImageRep** im, **HxSizes** sizes)

  Uniform filter.

7.274.1 Detailed Description

7.274.2 Function Documentation

7.274.2.1 HxImageRep L_HXIMAGEREP HxUniform (HxImageRep *im*, HxSizes *sizes*)

Uniform filter.

```
15 {
16     HxString fname("HxUniform");
17 
18     if (im.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23 
24     if ((sizes.x() < 1) || (sizes.x() > im.dimensionSize(1)))
25     {
26         HxGlobalError::instance()->reportError(fname, "illegal size x", HxGlobalError::HX_GE_INVALID);
27         return HxImageRep();
28     }
29 
30     if ((sizes.y() < 1) || (sizes.y() > im.dimensionSize(2)))
31     {
32         HxGlobalError::instance()->reportError(fname, "illegal size y", HxGlobalError::HX_GE_INVALID);
33         return HxImageRep();
34     }
35     if (im.dimensionality() == 3)
36     {
37         if ((sizes.z() < 1) || (sizes.z() > im.dimensionSize(3)))
38         {
39             HxGlobalError::instance()->reportError(fname, "illegal size z", HxGlobalError::HX_GE_INVALID);
40             return HxImageRep();
41         }
42     }
43 
44     // An image signature for a 2D image with 64-bit real valued scalar pixels
45     HxImageSignature sig(2, 1, REAL_VALUE, 64);
46 
47     // Construct the separable kernels
48     HxImageRep kx = HxImageFactory::instance().fromValue(sig,
```

```
49         HxSizes(sizes.x(),1,1), 1./sizes.x());
50     HxImageRep ky = HxImageFactory::instance().fromValue(sig,
51         HxSizes(sizes.y(),1,1), 1./sizes.y());
52     HxImageRep kz = HxImageFactory::instance().fromValue(sig,
53         HxSizes(sizes.z(),1,1), 1./sizes.z());
54
55     // and apply the operation
56     if (im.dimensionality() == 3) {
57         return im.genConv3dSep(kx, ky, kz, "mul", "addAssign",
58                                HxImageRep::ARITH_PREC);
59     }
60
61     return im.genConv2dSep(kx, ky, "mul", "addAssign", HxImageRep::ARITH_PREC);
62 }
```

7.275 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.

7.275.1 Detailed Description

7.275.2 Function Documentation

7.275.2.1 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(im.dimensionality(), 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 }
```

7.276 HxUpoSetPartImageInst.c File Reference

```
#include "HxUpoSetPartImage.h"
#include "HxImgFtorUpo.h"
#include "HxIncludedSigs.h"
```

```
#include "HxInstUpo.h"
```

Namespaces

- namespace **HxInstantiatorSetPartImage_c**

Compounds

- struct **HxInstantiatorSetPartImg**

  Instantiator for unary pixel operation.

Functions

- int **HxUpoSetPartImageInst** ()

  Dummy function to instantiate variables in this file.

7.276.1 Detailed Description

7.276.2 Function Documentation

7.276.2.1 int HxUpoSetPartImageInst ()

Dummy function to instantiate variables in this file.

```
64 {
65     int i = 3;
66     return i;
67 }
```

7.277 HxValleyRemoval.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxValleyRemoval** (**HxImageRep** im, int conn, int minarea)

7.277.1 Detailed Description

7.278 HxValueList.h File Reference

```
#include "HxValue.h"
#include <list>
```

Compounds

- class **HxValueList**
  Class definition for list of **HxValue** (p. 1253)*'s.*

Typedefs

- typedef HxValueList::iterator **HxValueListIter**
  Iterator for **HxValueList** (p. 1261).
- typedef HxValueList::const_iterator **HxValueListConstIter**
  Const iterator for **HxValueList** (p. 1261).
- typedef **HxValueList::back_insert_iterator HxValueListBackInserter**
  Back inserter for **HxValueList** (p. 1261).

7.278.1 Detailed Description

7.278.2 Typedef Documentation

7.278.2.1 typedef HxValueList::iterator HxValueListIter

Iterator for **HxValueList** (p. 1261).

7.278.2.2 typedef HxValueList::const_iterator HxValueListConstIter

Const iterator for **HxValueList** (p. 1261).

7.278.2.3 typedef HxValueList::back_insert_iterator HxValueListBackInserter

Back inserter for **HxValueList** (p. 1261).

7.279 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. 1253).

Functions

- std::ostream & **HxValueType_put** (std::ostream &os, **HxValueType** val)
- std::ostream & **operator<<** (std::ostream &os, **HxValueType** val)
- **HxString makeString** (**HxValueType** val)

7.279.1 Detailed Description

7.279.2 Enumeration Type Documentation

7.279.2.1 enum HxValueType

The type of a **HxValue** (p. 1253).

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 };
```

7.280 HxVec2Byte.h File Reference

```
#include "HxVec2Tem.h"
#include "HxByte.h"
```

Typedefs

- typedef **HxVec2Tem**< **HxByte** > **HxVec2Byte**

  Type definition for vector of 2 HxByte's (unsigned chars).

Variables

- **HxVec2Byte dummyVec2Byte**

7.280.1 Detailed Description

7.280.2 Typedef Documentation

7.280.2.1 typedef HxVec2Tem<HxByte> HxVec2Byte

Type definition for vector of 2 HxByte's (unsigned chars).

7.281 HxVec2Float.h File Reference

```
#include "HxVec2Tem.h"
```

Typedefs

- typedef **HxVec2Tem**< float > **HxVec2Float**
  Type definition for vector of 2 float's.

Variables

- **HxVec2Float dummyVec2Float**

7.281.1 Detailed Description

7.281.2 Typedef Documentation

7.281.2.1 typedef HxVec2Tem<float> HxVec2Float

Type definition for vector of 2 float's.

7.282 HxVec2Short.h File Reference

```
#include "HxVec2Tem.h"
```

Typedefs

- typedef **HxVec2Tem**< short > **HxVec2Short**
  Type definition for vector of 2 shorts.

Variables

- **HxVec2Short dummyVec2Short**

7.282.1 Detailed Description

7.282.2 Typedef Documentation

7.282.2.1 typedef HxVec2Tem<short> HxVec2Short

Type definition for vector of 2 shorts.

7.283 HxVec3Byte.h File Reference

```
#include "HxVec3Tem.h"
#include "HxByte.h"
```

Typedefs

- typedef **HxVec3Tem**< **HxByte** > **HxVec3Byte**
  Type definition for vector of 3 HxByte's (unsigned chars).

Variables

- **HxVec3Byte dummyVec3Byte**

7.283.1 Detailed Description

7.283.2 Typedef Documentation

7.283.2.1 typedef HxVec3Tem<HxByte> HxVec3Byte

Type definition for vector of 3 HxByte's (unsigned chars).

7.284 HxVec3Float.h File Reference

```
#include "HxVec3Tem.h"
```

Typedefs

- typedef **HxVec3Tem**< float > **HxVec3Float**
  Type definition for vector of 3 floats.

Variables

- **HxVec3Float dummyVec3Float**

7.284.1 Detailed Description

7.284.2 Typedef Documentation

7.284.2.1 typedef HxVec3Tem<float> HxVec3Float

Type definition for vector of 3 floats.

7.285 HxVec3Short.h File Reference

```
#include "HxVec3Tem.h"
```

Typedefs

- typedef **HxVec3Tem**< short > **HxVec3Short**

  Type definition for vector of 3 shorts.

Variables

- **HxVec3Short dummyVec3Short**

7.285.1 Detailed Description

7.285.2 Typedef Documentation

7.285.2.1 typedef HxVec3Tem<short> HxVec3Short

Type definition for vector of 3 shorts.

7.286 HxWatershed.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxWatershed** (**HxImageRep** input, int conn)

  Watershed function creates the result image by detecting the domain of the catchment basins of "im" using the "flooding definition", according to the connectivity defined by "sf".

7.286.1 Detailed Description

7.286.2 Function Documentation

7.286.2.1 HxImageRep L_HXIMAGEREP HxWatershed (HxImageRep *input*, int *conn*)

Watershed function creates the result image by detecting the domain of the catchment basins of "im" using the "flooding definition", according to the connectivity defined by "sf".

According to the flag "linereg" the result image will be a labeled image of the catchment basins domain or just a binary image that presents the watershed lines.

Implementation based on Vincent algorithm (PAMI) Evaluation using SDC Matlab toolbox (www.morph.com)

- `im` is the input image (**HxImageRep** (p. 620))
- `mark` is the marker image (**HxImageRep** (p. 620))
- `sf` is the structuring function; default type is crossSF (**HxSF** (p. 1186))
- `linereg` HxString. Possible values 'LINES' or ' REGIONS'. Default: "LINES".

Returns:
the watershed Image, ImageRep

∗ references: L. Vincent and P. Soille, Watersheds in digital spaces: an efficient algorithm based on immersion simulations, IEEE Transactions on Pattern Analysis and Machine Intelligence. 13:583-598, 1991.

Remarks:
this queue based implementation is based on sorted lists of pixels for color images there is no partial ordering therefore we cannot use this for color

```
433 {
434     HxTagList tags;
435     HxAddTag(tags, "connectivity", conn);
436     HxImageRep mask=input;
437     //to speedup compute here the min and max gray values
438     int     hmin,hmax;
439     hmin = HxPixMin(input).HxScalarIntValue().x();
440     hmax = HxPixMax(input).HxScalarIntValue().x();
441     HxAddTag(tags,"hmin",hmin);
442     HxAddTag(tags,"hmax",hmax);
443
444     //for more speedup, use a histogram to indicate the existing gray levels
445
446     HxImageRep out = input.queueBasedOp(mask, "qWaterShedLV", tags);
447
448     //because the LucVincent algorithm doesn't get all the watershed points
449     //this neigborhoud operator replaces the smallest label with wshed value
450     HxAddTag(tags, "conn", conn);
451     HxAddTag(tags, "wshedval", WSHEDVAL);
452
453     out = out.neighbourhoodOp("lwshed", tags);
454
455
456     return out;
457 }
```

7.287 HxWatershedMarkers.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxWatershedMarkers** (**HxImageRep** input, **HxImageRep** mask, int conn=4, bool doLabelMask=false)

  Watershed from markers.

7.287.1 Detailed Description

7.287.2 Function Documentation

7.287.2.1 HxImageRep L_HXIMAGEREP HxWatershedMarkers (HxImageRep *input*, HxImageRep *mask*, int *conn* = 4, bool *doLabelMask* = false)

Watershed from markers.

- `im` is the input image (**HxImageRep** (p. 620))
- `mask` is the marker image (**HxImageRep** (p. 620))
- `conn` is the connectivity (4 or 8)
- `doLabelMask` (bool) indicate whether the marker images needs to be labeled before, or not

Returns:
the watershed Image, ImageRep

Remarks:
this version of watershed does not consider a cost function. See HxWatershedMarker2, for a cost-based implementation

R. Lotufo and A. Falcao. "The ordered queue and the optimality of the watershed approaches." In Mathematical Morphology and its Application to Image and Signal Processing, pages 341-350, Dordrecht, The Netherlands, 2000.

```
200 {
201     HxTagList tags;
202     HxAddTag(tags, "connectivity", conn);
203     HxAddTag(tags, "doLabelPlusPlus", doLabelMask);
204
205     return input.queueBasedOp(mask, "qWatershedMarkers", tags);
206 }
```

7.288 HxWatershedMarkers2.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxWatershedMarkers2** (**HxImageRep** input, **HxImageRep** mask, int conn=4, bool doLabelMask=false, int costMethod=0)

  Watershed from markers, second implementation.

7.288.1 Detailed Description

7.288.2 Function Documentation

7.288.2.1 HxImageRep L_HXIMAGEREP HxWatershedMarkers2 (HxImageRep *input*, HxImageRep *mask*, int *conn* = 4, bool *doLabelMask* = false, int *costMethod* = 0)

Watershed from markers, second implementation.

- `im` is the input image (**HxImageRep** (p. 620))
- `mask` is the marker image (**HxImageRep** (p. 620))
- `conn` is the connectivity (4 or 8)
- `doLabelMask` (bool) indicate whether the marker images needs to be labeled before, or not
- `costMethod` (int) indicate the cost function to be used in comparing to pixels: costMethod = 0: => w=neighbor.img costMethod = 1: => w=(center.img-neighbor.img)

R. Lotufo and A. Falcao. The ordered queue and the optimality of the watershed approaches. In Mathematical Morphology and its Application to Image and Signal Processing, pages 341{350, Dordrecht, The Netherlands, 2000.

```
256 {
257     HxTagList tags;
258     HxAddTag(tags, "connectivity", conn);
259     HxAddTag(tags, "doLabelPlusPlus", doLabelMask);
260     HxAddTag(tags, "costmethod", costMethod);
261 
262     return input.queueBasedOp(mask, "qWatershedMarkers2", tags);
263 }
```

7.289 HxWatershedSlow.h File Reference

```
#include "HxSF.h"
```

Functions

- **HxImageRep** L_HXIMAGEREP **HxWatershedSlow** (**HxImageRep** im, **HxSF** sf, **HxString** linereg)

  Watershed function creates the result image by detecting the domain of the catchment basins of ”im” indicated by the ”immersion definition”, according to the connectivity defined by ”sf”.

7.289.1 Detailed Description

7.289.2 Function Documentation

7.289.2.1 HxImageRep L_HXIMAGEREP HxWatershedSlow (HxImageRep *im*, HxSF *sf*, HxString *linereg*)

Watershed function creates the result image by detecting the domain of the catchment basins of ”im” indicated by the ”immersion definition”, according to the connectivity defined by ”sf”.

According to the flag ”linereg” the result image will be a labeled image of the catchment basins domain or just a binary image that presents the watershed lines.

Implementation based on Vincent algorithm (PAMI) Evaluation using SDC Matlab toolbox (www.morph.com)

- `im` is the input image (**HxImageRep** (p. 620))
- `mark` is the marker image (**HxImageRep** (p. 620))
- `sf` is the structuring function; default type is crossSF (**HxSF** (p. 1186))
- `linereg` HxString. Possible values ’LINES’ or ’ REGIONS’. Default: ”LINES”.

Returns:
the watershed Image, ImageRep

∗ references: L. Vincent and P. Soille, Watersheds in digital spaces: an efficient algorithm based on immersion simulations, IEEE Transactions on Pattern Analysis and Machine Intelligence. 13:583-598, 1991.

Remarks:

this queue based implementation is based on sorted lists of pixels for color images there is no partial ordering therefore we cannot use this for color

```
247 {
248     HxSizes             sz  = im.sizes();
249     HxImageSignature    sig = im.signature();
250 
251 //because there is no ordering in 3D space of color images, here we consider gray level images only
252 //
253     if( sig.pixelDimensionality() == 3 )
254     {
255         im = HxRGB2Intensity(im);
256         im = HxImageAsInt(im);
257 
258         HX_COUT << "this function is not implemented for color images!" << STD_ENDL;
259     }
260 
261 
262     int conn = sf.getConnectivity();
263 
264     HxImageSignature    sigOut; //default constructor, make a singature 2DINT
265     //initialize the resltImage with -1
266     HxImageRep      im0 = HxMakeFromValue(sigOut, sz, -1);
267     HxImageRep      imd = HxMakeFromValue(sigOut, sz, 0);
268 
269     int     currentLabel    = 0;
270     int     currentDist;
271 
272     HxPoint2 p;
273     HxPoint2 p1(-1,-1);
274     HxImgAsListIterator p1lIter, p2lIter;
275 
276 //fifo
277     std::list< HxPoint2>  fifo;
278 
279 
280     int     mask    = -2;   //the initial value of a threshold level
281     int     wshed   = -3; //0;  //value of pixels belonging to the watersheds
282     int     h, hmin,hmax;
283     hmin = HxPixMin(im).HxScalarIntValue().x(); //what about color image?
284     hmax = HxPixMax(im).HxScalarIntValue().x();
285 
286 //sort the pixels in im in the increasing order of their gray values
287 //use HxGetPoints,  HxGetValues, and sort these lists
288 
289     HxValueList        vl;
290     HxPointList        pl;
291     HxPointListIter    plIter;
292     HxValueListIter    vlIter;
293 
295 HxTimer timer;
296 timer.start();
297 
298     HxGetPoints(im, std::back_inserter(pl));
299     HxGetValues(im, pl.begin(), pl.end(), std::back_inserter(vl));
300 
301 
302     HxImgAsList iml;
303 
304 
305 //  printf("\nSTART imlsize=%d\n", iml.size());
306 
307     for(plIter = pl.begin(), vlIter = vl.begin();
308         plIter!= pl.end() && vlIter!= vl.end();
```

```
309         plIter++, vlIter++)
310         {
311             HxPointAndValue pv((*plIter), (*vlIter).HxScalarIntValue());
312             iml.push_back(pv);
313         }
314
315     //this is not possible because HxValue does not implement the < operator
316     //vl.sort();
317 //  printf("\nimlsize=%d\n", iml.size());
318
319 //  iml.sort();
320     //sort the vector, not the list
321     std::sort(iml.begin(), iml.end());
322
323     //printf("\n");
324 timer.stop();
325 show(timer);
326 timer.start();
327
328 //  printf("\nimlsize=%d\n", iml.size());
329 //  printf("hmin=%d hmax=%d\n", hmin, hmax);
330
331     HxImgAsListIterator imlIter,imlIter2, startIter, startIterMin ;
332
333
334     startIter = iml.begin();
335     startIterMin = iml.begin();
336 //<<1>>
337     for(h = hmin; h<=hmax; h++)
338     {
339         bool hIsNotPresent = false;
340         bool processed = false;
341         //geodesic SKIZ of level h-1 inside level h
342         for(imlIter=startIter; imlIter!=iml.end(); imlIter++)
343         {
344             if( (*imlIter).v == h)  //LT here you can speedup using a set instead of list of values
345             {
346                 processed = true;
347
348                 p = (*imlIter).p;
349                 //im0(p) <- mask
350                 im0.setAt(mask, p.x(), p.y() );
351
352                 //now check the vecinity of pixel p in im0
353                 if( HxCompareNeighborPixels(im0, p, conn, GT, 0) ||  //if one of the nighbor is already labeled.
354                     HxCompareNeighborPixels(im0, p, conn, EQ, wshed) )
355                 {
356                     imd.setAt(1, p.x(), p.y() );
357                     //and put 'p' in fifo list
358                     fifo.push_back(p);
359                 }
360
361             }
362             else    //because the image is sorted by pixelvalues, you can speed up by breaking this loop
363             {
364                 if((*imlIter).v > h && !processed) //this level doesn't exist in the image, so break the main loop
365                 {
366                     hIsNotPresent = true;
367                     break;
368                 }
369                 if(processed)
370                 {
371                     startIter = imlIter;
372                     break;
373                 }
```

```
374             }
375
376         }
377
378     if(hIsNotPresent)
379         continue;
380 // printf("h=%3d: fifo size=%7d\n",h, fifo.size());
381 //<<2>>
382     currentDist=1;
383     HxPoint2 fictionPoint(-1,-1);
384     fifo.push_back(fictionPoint);
385     while(true)
386     {
387         p = *fifo.begin();
388         fifo.erase(fifo.begin());
389         if ( p==fictionPoint)
390         {
391             if(fifo.empty())
392                 break;
393             else
394             {
395                 fifo.push_back(fictionPoint);
396                 currentDist++;
397                 p = *fifo.begin();
398                 fifo.erase(fifo.begin());
399             }
400         }
401 //      printf("currentDist=%d\n",currentDist);
402 //<<3>>
403
404         //hey! what if point p is not correct?
405         HxImgAsList p1l=HxGetNeighborPixels(im0, p, conn);
406 //      if(!p1l.empty())
407         for(p1lIter=p1l.begin(); p1lIter!=p1l.end(); p1lIter++)
408         {
409             p1 = (*p1lIter).p;
410             if( (imd.getAt(p1.x(),p1.y()).HxScalarIntValue() < currentDist ) &&
411                 ((im0.getAt(p1.x(),p1.y()).HxScalarIntValue() > 0 )          ||
412                 (im0.getAt(p1.x(),p1.y()).HxScalarIntValue() == wshed )) )
413             {
414                 //i.e. p' belongs to an already labeled basin or to the watersheds
415                 //then p get the value of p1 or 'wshed' value
416
417                 if( im0.getAt(p1.x(),p1.y()).HxScalarIntValue() > 0  ) //a basin point
418                 {
419                     if( (im0.getAt(p.x(),p.y()).HxScalarIntValue() == mask) ||
420                         (im0.getAt(p.x(),p.y()).HxScalarIntValue() == wshed) )
421                         im0.setAt(im0.getAt(p1.x(),p1.y()).HxScalarIntValue(),p.x(),p.y());
422                     else
423                         if( im0.getAt(p.x(),p.y()).HxScalarIntValue() != im0.getAt(p1.x(),p1.y()).HxScalarIntValue(
424                         im0.setAt(wshed,p.x(),p.y());
425                 }
426                 else
427                     //i.e. p' is a wshed point
428                     //then check the value of p
429                     if(im0.getAt(p.x(),p.y()).HxScalarIntValue() == mask )
430                                     im0.setAt(wshed,p.x(),p.y());
431             }
432             else
433                 if( (im0.getAt(p1.x(),p1.y()).HxScalarIntValue() == mask) &&
434                     (imd.getAt(p1.x(),p1.y()).HxScalarIntValue() == 0) )
435                 {
436                     imd.setAt(currentDist+1,p1.x(),p1.y());
437                     fifo.push_back(p1);
438                 }
```

```
439         }
440     }//end while(true) -> endless loop; break when fifo is empty
441
442
443     //lt for debug
444 //export the distance image
445 //  HxImageRep imdE = HxImageAsByte(imd);
446 //  HxWriteFile(imdE, "dist.tif");
447
448
449
450 //<<4>>
451
452       // check if new minima have been discovered
453
454         bool minFound=false;
455         for(imlIter2 = startIterMin; imlIter2 != iml.end(); imlIter2++)
456         {
457             if( (*imlIter2).v.x() == h)
458             {
459                 minFound = true;
460                 p = (*imlIter2).p;
461                 //the distance associated with p is reset to 0
462                 imd.setAt(0, p.x(),p.y());
463                 if(im0.getAt(p.x(),p.y()).HxScalarIntValue().x() == mask )
464                 {
465                     currentLabel++;
466                     fifo.push_back(p);
467                     im0.setAt(currentLabel, p.x(),p.y());
468                     while(!fifo.empty())
469                     {
470                         p1 = *fifo.begin();
471                         fifo.erase(fifo.begin());
472                         HxPoint2 p2;
473                         HxImgAsList p2l = HxGetNeighborPixels(im0, p1, conn);
474                         for(p2lIter=p2l.begin(); p2lIter!=p2l.end(); p2lIter++)
475                         {
476                             p2 = (*p2lIter).p;
477                             if(im0.getAt(p2.x(),p2.y()).HxScalarIntValue().x() == mask)
478                             {
479                                 fifo.push_back(p2);
480                                 im0.setAt(currentLabel, p2.x(),p2.y());
481                             }
482                         }
483                     }
484                 }
485             }
486             else
487             {
488                 if(minFound)
489                 {
490                     startIterMin=imlIter2;
491                     break;
492                 }
493             }
494         }
495
496 //  HxImageRep im0E = HxImageAsByte(im0);
497 //  HxWriteFile(im0E, "im0before4.tif");
498
499     }//end for every h level in [hmin,hmax]
500
501 timer.stop();
502 show(timer);
503
```

```
504 /*************************/
505 //end lucVincent algorithm, now make the tesselation image
506 /************************/
507
508 //implement this with neighbourhood operators
509
510 timer.start();
511
512     HxImageRep      imres = im0;
513
514
515     pl.clear();
516     vl.clear();
517
518     //take care HxGetPoints doesn't include the zero pixels!
519     //but here I changed whsed value from 0 to -3
520     HxGetPoints(im0, std::back_inserter(pl));
521     HxGetValues(im0, pl.begin(), pl.end(), std::back_inserter(vl));
522
523 //  printf("plsize=%d, vlsize=%d\n",pl.size(), vl.size());
524  STD_COUT << "imres : " << imres.signature() << ", sizes: " << imres.sizes() << STD_ENDL;
525
526     for(plIter = pl.begin(), vlIter = vl.begin();
527         plIter!= pl.end() && vlIter!= vl.end();
528         plIter++, vlIter++)
529         {
530             HxPoint2 pc = (*plIter);
531             int v = (*vlIter).HxScalarIntValue().x();
532
533             if( v == wshed)
534             {
535                 //leon it is better to give to wshed pixel the label of
536                 //the smallest label among neighbors - this is just to be consistent
537                 HxImgAsList p0l=HxGetNeighborPixels(im0, pc, conn);
538
539                 int maxValNg = 0;
540                 for(p1lIter=p0l.begin(); p1lIter!=p0l.end(); p1lIter++)
541                 {
542                     if(( *p1lIter).v.x() > maxValNg)
543                         maxValNg = (*p1lIter).v.x();
544                 }
545
546 STD_COUT << "    in loop before setAt" << STD_ENDL;
547                 if(maxValNg > 0 )
548                     {
549 STD_COUT << "values : " << maxValNg << "," << pc.x() << "," << pc.y() << STD_ENDL;
550                         HxValue maxValNgV(maxValNg);
551                         int x = pc.x();
552                         int y = pc.y();
553 STD_COUT << "values local : " << maxValNgV << "," << x << "," << y << STD_ENDL;
554                         HxValue orgV = imres.getAt(x, y);
555 STD_COUT << "org value : " << orgV << STD_ENDL;
556                         //imres.setAt(maxValNg, pc.x(),pc.y());
557                         //imres.setAt(maxValNgV, x, y);
558                         imres.setAt(orgV, x, y);
559                     }
560 STD_COUT << "    in loop after setAt" << STD_ENDL;
561
562             }
563         }
564
565
566
567
568
```

```
569     im0 = imres;
570
571 //to separate catchment basines
572     if(linereg=="LINES"){
573
574         //im0 =  HxLWshed(im0,conn,wshed);
575         HxTagList tags;
576         HxAddTag(tags, "conn", conn);
577         HxAddTag(tags, "wshedval", wshed);
578         im0 = im0.neighbourhoodOp("lwshed", tags);
579
580 //this can be implemented iwth LUT, some  unary pixel operation anyway...
581         pl.clear();
582         vl.clear();
583         HxGetPoints(im0, std::back_inserter(pl));
584         HxGetValues(im0, pl.begin(), pl.end(), std::back_inserter(vl));
585
586         for(plIter = pl.begin(), vlIter = vl.begin();
587             plIter!= pl.end() && vlIter!= vl.end();
588             plIter++, vlIter++)
589             {
590                 HxPoint2 pc = (*plIter);
591                 int v = (*vlIter).HxScalarIntValue().x();
592                 if(v == wshed )
593                         imres.setAt(255, pc.x(),pc.y());
594                 else
595                         imres.setAt(0, pc.x(),pc.y());
596             }
597
598         im0=imres;
599
600     }
601
602     if(linereg=="REGIONS"){
603
604 //implemented with neighbourhood operator
605         //im0 =  HxLWshed(im0,conn,wshed);
606         HxTagList tags;
607         HxAddTag(tags, "conn", conn);
608         HxAddTag(tags, "wshedval", wshed);
609         im0 = im0.neighbourhoodOp("lwshed", tags);
610
611
612         pl.clear();
613         vl.clear();
614         HxGetPoints(im0, std::back_inserter(pl));
615         HxGetValues(im0, pl.begin(), pl.end(), std::back_inserter(vl));
616
617         for(plIter = pl.begin(), vlIter = vl.begin();
618             plIter!= pl.end() && vlIter!= vl.end();
619             plIter++, vlIter++)
620             {
621                 HxPoint2 pc = (*plIter);
622                 int v = (*vlIter).HxScalarIntValue().x();
623                 if(v == wshed )
624                         imres.setAt(0, pc.x(),pc.y());
625             }
626
627         im0=imres;
628
629     }
630
631 timer.stop();
632 show(timer);
633
```

```
634     return im0;
635 }
```

7.290 HxWeightMaskSum.h File Reference

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

Functions

- **HxValue** L_HXIMAGEREP **HxWeightMaskSum** (**HxImageRep** im, **HxImageRep** mask, **HxPoint** p)

  Compute the sum of all pixels in "im" weighed by "mask".

7.290.1 Detailed Description

7.290.2 Function Documentation

7.290.2.1 HxValue L_HXIMAGEREP HxWeightMaskSum (HxImageRep *im*, HxImageRep *mask*, HxPoint *p*)

Compute the sum of all pixels in "im" weighed by "mask".

The function considers all pixels within the area starting at point "p" with a size equal to that of "mask".

Implementation specifics :

- Pattern : **Export operation with an extra image** (p. 38)
- Variation : **Translation invariant, 1 phase export operation with an extra image** (p. 38)
- The pixel functor : **HxExportExtraWeightMaskSum** (p. 546)
- Instantiations : HxExportExtraWeightMaskSum_c

```
13 {
14     HxBoundingBox bb(mask.sizes());
15     HxTagList tags;
16     HxAddTag(tags, "boundingBox", bb);
17     HxAddTag(tags, "doOffsetExtra", false);
18     im.exportOpExtra("weightMaskSum", mask, tags);
19     HxValue v = HxGetTag<HxValue>(tags, "result");
20     return v;
21 }
```

7.291 HxWriteFile.h File Reference

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

Functions

- bool L_HXIMAGEREP **HxWriteFile** (**HxImageRep** im, **HxString** fileName)

  Write image to file using HxImgFileIo.

7.291.1 Detailed Description

7.291.2 Function Documentation

7.291.2.1 bool L_HXIMAGEREP HxWriteFile (HxImageRep *im*, HxString *fileName*)

Write image to file using HxImgFileIo.

```
19 {
20     HxString fname("HxWriteFile");
21
22     if (im.isNull())
23     {
24         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
25         return false;
26     }
27
28     HxString extName = HxSystem::instance().extName(fileName);
29
30     if ((extName == "jpg") || (extName == "JPG") || (extName == "jpeg") ||
31         (extName == "JPEG") || (extName == "jfif") || (extName == "JFIF"))
32     {
33         if ((im.signature().pixelType() == INT_VALUE) &&
34             (im.signature().pixelPrecision() != 8))
35         {
36             HxGlobalError::instance()->reportError(fname, "jpeg format supported only for byte precision", HxGlobalEr
37             return false;
38         }
39         if ((im.signature().pixelType() == INT_VALUE) &&
40             (im.pixelDimensionality() != 1) && (im.pixelDimensionality() != 3))
41         {
42             HxGlobalError::instance()->reportError(fname, "jpeg format supported only 1D and 3D pixels", HxGlobalErro
43             return false;
44         }
45         if (im.signature().pixelType() != INT_VALUE)
46         {
47             HxGlobalError::instance()->reportError(fname, "jpeg format supported only for integer values, byte precis
48             return false;
49         }
50     }
51     if ((extName == "png") || (extName == "PNG") || (extName == "ping") ||
52         (extName == "PING"))
53     {
54         if ((im.signature().pixelType() == INT_VALUE) &&
55             (im.pixelDimensionality() != 1) && (im.pixelDimensionality() != 3))
56         {
57             HxGlobalError::instance()->reportError(fname, "PNG format supported only 1D and 3D pixels", HxGlobalError
58             return false;
59         }
60     }
61
62     HxImgFileReader* reader = HxImgFileIoTable::instance().getReader(extName);
63
64     if (!reader)
65     {
66         HxString errstr = HxString("Illegal file type '") + extName + HxString("'");
67         HxGlobalError::instance()->reportError(fname, im.name(), errstr, HxGlobalError::HX_GE_INVALID);
68         return false;
69     }
70
71     HxTagList tags;
72     return HxImageFactory::instance().writeFile(im, fileName, tags);
73 }
```

7.292 HxXor.h File Reference

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

Functions

- **HxImageRep** L_HXIMAGEREP **HxXor** (**HxImageRep** im1, **HxImageRep** im2)
  Exclusive or.

7.292.1 Detailed Description

7.292.2 Function Documentation

7.292.2.1 HxImageRep L_HXIMAGEREP HxXor (HxImageRep *im1*, HxImageRep *im2*)

Exclusive or.

The function performs exclusive or (see **Pixels** (p. 3)) on all pixels in the input images via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoXor** (p. 469). The image functor instantiator : **HxInstantiatorXor** (p. 927).

```
15 {
16     HxString fname("HxXor");
17 
18     if (im1.isNull())
19     {
20         HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVALID);
21         return HxImageRep();
22     }
23     if (im2.isNull())
24     {
25         HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVALID);
26         return HxImageRep();
27     }
28 
29     if (im1.signature().imageDimensionality() != im2.signature().imageDimensionality())
30     {
31         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
32         return HxImageRep();
33     }
34     if (im1.signature().pixelDimensionality() != im2.signature().pixelDimensionality())
35     {
36         HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
37         return HxImageRep();
38     }
39     if (im1.signature().pixelDimensionality() != 1)
40     {
41         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
42         return HxImageRep();
43     }
44 
45     if (im1.signature().pixelType() != im2.signature().pixelType())
46     {
47         HxGlobalError::instance()->reportError(fname, "unequal pixel types", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
48         return HxImageRep();
```

```
49     }
50     if (im1.signature().pixelType() != INT_VALUE)
51     {
52         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer values", HxGlobalE
53         return HxImageRep();
54     }
55
56     if (im1.signature().pixelPrecision() != im2.signature().pixelPrecision())
57     {
58         HxGlobalError::instance()->reportError(fname, "unequal pixel precisions", HxGlobalError::HX_GE_UNEQUAL_IMAGES
59         return HxImageRep();
60     }
61
62     if (im1.sizes().x() != im2.sizes().x())
63     {
64         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
65         return HxImageRep();
66     }
67     if (im1.sizes().y() != im2.sizes().y())
68     {
69         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
70         return HxImageRep();
71     }
72     if (im1.signature().imageDimensionality() > 2)
73     {
74         if (im1.sizes().z() != im2.sizes().z())
75         {
76             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
77             return HxImageRep();
78         }
79     }
80
81     return im1.binaryPixOp(im2, "xor");
82 }
```

7.293 HxXorVal.h File Reference

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

Functions

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

  Exclusive or.

7.293.1 Detailed Description

7.293.2 Function Documentation

7.293.2.1 HxImageRep L_HXIMAGEREP HxXorVal (HxImageRep *im*, HxValue *val*)

Exclusive or.

The function performs exclusive or (see **Pixels** (p. 3)) on all pixels in the input image via a binary pixel operation (see **Images** (p. 8)).

Implementation specifics : The pixel functor : **HxBpoXor** (p. 469). The image functor instantiator : **HxInstantiatorXorV** (p. 928).

```
13 {
14     HxString fname("HxXorVal");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21 
22     if (im.signature().pixelDimensionality() != 1)
23     {
24         HxGlobalError::instance()->reportError(fname, "logical operators are only valid for scalar images", HxGlobalE
25         return HxImageRep();
26     }
27 
28     if (im.signature().pixelType() != INT_VALUE)
29     {
30         HxGlobalError::instance()->reportError(fname, "logical operators are only valid on integer images", HxGlobalE
31         return HxImageRep();
32     }
33 
34     if (val.tag() != HxValue::SI)
35     {
36         HxGlobalError::instance()->reportError(fname, "only scalar integer value supported", HxGlobalError::HX_GE_INV
37         return HxImageRep();
38     }
39 
40     return im.binaryPixOp(val, "xor");
41 }
```

Chapter 8

Class Reference

8.1 DeviceEnumerator Class Reference

Enumerates system devices (that can be used as filters).

`#include <DeviceEnumerator.h>`

Public Methods

- **DeviceEnumerator** (REFCLSID devClass=CLSID_VideoInputDeviceCategory)
- ~**DeviceEnumerator** ()
- char ** **getNames** ()
- HRESULT **getFilter** (int num, IBaseFilter **filter)
- HRESULT **getFilter** (const char *name, IBaseFilter **filter)

8.1.1 Detailed Description

Enumerates system devices (that can be used as filters).

devClass can be (among others) :

- CLSID_VideoInputDeviceCategory: Video capture category.
- CLSID_LegacyAmFilterCategory: Filter category.
- CLSID_VideoCompressorCategory: Video compressor category.
- CLSID_AudioCompressorCategory: Audio compressor category.
- CLSID_AudioInputDeviceCategory: Audio source category.
- CLSID_AudioRendererCategory: Audio renderer category.
- CLSID_MidiRendererCategory: Midi renderer category.

The documentation for this class was generated from the following files:

- **DeviceEnumerator.h**
- DeviceEnumerator.c

8.2 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.

8.2.1 Detailed Description

Class definition for arrows in R2.

An arrow has a position (origin) and a direction (displace). Both have real-value coordinates (R2).

8.2.2 Constructor & Destructor Documentation

8.2.2.1 HxArrowR2::HxArrowR2 () `[inline]`

Constructor.

```
60 {
61 }
```

8.2.2.2 HxArrowR2::HxArrowR2 (const HxVectorR2 & *v*) [inline]

Construct from vector (origin = (0,0)).

```
64                                          : _origin(0, 0), _displace(v)
65 {
66 }
```

8.2.2.3 HxArrowR2::HxArrowR2 (const HxPointR2 & *p*, const HxVectorR2 & *v*) [inline]

Construct from given point and vector.

```
69                                                                  : _origin(p),
70                                                                      _displace(v)
71 {
72 }
```

8.2.2.4 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 }
```

8.2.2.5 HxArrowR2::~HxArrowR2 () [inline]

Destructor.

```
82 {
83 }
```

8.2.3 Member Function Documentation

8.2.3.1 HxPointR2 HxArrowR2::origin () [inline]

Get the origin.

```
87 {
88     return _origin;
89 }
```

8.2.3.2 HxVectorR2 HxArrowR2::displace () [inline]

Get the displacement.

```
93 {
94     return _displace;
95 }
```

8.2.3.3 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**

8.3 HxBlob2d Class Reference

Class definition for a blob in a 2D image.

`#include <HxBlob2d.h>`

Public Methods

- **HxBlob2d** ()
  Constructor.
- **HxBlob2d** (int label, int xmin, int ymin, int width, int height)
  Constructor.
- ~**HxBlob2d** ()
  Destructor.
- int **ident** () const
  Get the identifier of this blob.
- int **getLabel** () const
  Get the label of this blob (unique only within the labeled image.
- **HxPoint startMaer** () const
  Get the upper left corner of the MAER.
- **HxSizes sizeMaer** () const
  Get the sizes of the MAER.
- template<class T> void **addFeature** (**HxString** name, const T &val)
  Add combination name,val to the list of features.
- template<class T> void **addFeature** (**HxNameTable::sizeType** id, const T &val)
  Add combination id,val to the list of features.
- template<class T> bool **getFeature** (**HxString** name, T &val) const
  Get feature idetified by name from the list.

- template<class T> bool **getFeature** (**HxNameTable::sizeType** id, T &val) const

  Get feature idetified by name from the list.

- int **getFeatureInt** (**HxString** name) const

  Wrapper around getFeature for int's.

- **HxValue getFeatureValue** (**HxString** name) const

  Wrapper around getFeature for **HxValue** (p. 1253)*'s.*

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Put blob on stream.

Static Public Methods

- std::vector< **HxString** > **getFeatureNames** ()

  Get all feature names.

8.3.1 Detailed Description

Class definition for a blob in a 2D image.

Basically, a blob is a label in an identification image. In other words, a blob is a set of pixels that are "connected".

8.3.2 Constructor & Destructor Documentation

8.3.2.1 HxBlob2d::HxBlob2d ()

Constructor.

```
24 {
25     _ident = _nr++;
26 }
```

8.3.2.2 HxBlob2d::HxBlob2d (int *label*, int *xmin*, int *ymin*, int *width*, int *height*)

Constructor.

```
41 {
42     _ident = _nr++;
43     _label = label;
44     _xmin = xmin;
45     _ymin = ymin;
46     _width = width;
47     _height = height;
48 }
```

8.3.2.3 HxBlob2d::~HxBlob2d ()

Destructor.

```
51 {
52 }
```

8.3.3 Member Function Documentation

8.3.3.1 int HxBlob2d::ident () const

Get the identifier of this blob.

```
56 {
57     return _ident;
58 }
```

8.3.3.2 int HxBlob2d::getLabel () const

Get the label of this blob (unique only within the labeled image.

```
76 {
77     return _label;
78 }
```

8.3.3.3 HxPoint HxBlob2d::startMaer () const

Get the upper left corner of the MAER.

```
82 {
83     return HxPoint(_xmin, _ymin, 0);
84 }
```

8.3.3.4 HxSizes HxBlob2d::sizeMaer () const

Get the sizes of the MAER.

```
88 {
89     return HxSizes(_width, _height, 1);
90 }
```

8.3.3.5 template<class T> void HxBlob2d::addFeature (HxString *name*, const T & *val*) [inline]

Add combination name,val to the list of features.

```
77     {
78         HxNameTable::sizeType id = _featNameTable.getId(name);
79         addFeature(id, val);
80     }
```

8.3.3.6 template<class T> void HxBlob2d::addFeature (HxNameTable::sizeType *id*, const T & *val*) [inline]

Add combination id,val to the list of features.

```
85     {
86         HxBlob2dFeature* feat = new HxBlob2dFeatureTem<T>(val);
87         _features[id] = feat;
88     }
```

8.3.3.7 template<class T> bool HxBlob2d::getFeature (HxString *name*, T & *val*) const [inline]

Get feature idetified by name from the list.

```
93     {
94         HxNameTable::sizeType id = _featNameTable.getId(name);
95         return getFeature(id, val);
96     }
```

8.3.3.8 template<class T> bool HxBlob2d::getFeature (HxNameTable::sizeType *id*, T & *val*) const [inline]

Get feature idetified by name from the list.

```
101     {
102         FeatureMap::const_iterator it = _features.find(id);
103         if (it == _features.end())
104             return false;
105         HxBlob2dFeature* base = (*it).second;
106         HxBlob2dFeatureTem<T>* featPtr =
107             dynamic_cast<HxBlob2dFeatureTem<T>* > (base);
108         if (featPtr == 0)
109             return false; // was other type
110         val = featPtr->getValue();
111         return true;
112     }
```

8.3.3.9 int HxBlob2d::getFeatureInt (HxString *name*) const

Wrapper around getFeature for int's.

```
144 {
145     int val;
146     getFeature(name, val);
147     return val;
148 }
```

8.3.3.10 HxValue HxBlob2d::getFeatureValue (HxString *name*) const

Wrapper around getFeature for **HxValue** (p. 1253)'s.

```
152 {
153     HxValue val;
154     getFeature(name, val);
155     return val;
156 }
```

8.3.3.11 STD_OSTREAM & HxBlob2d::put (STD_OSTREAM & *os*) const

Put blob on stream.

```
160 {
161     os << "Blob " << ident() << ", start: " << startMaer() << ", size: "
162        << sizeMaer() << STD_ENDL;
163     return os;
164 }
```

8.3.3.12 std::vector< HxString > HxBlob2d::getFeatureNames () [static]

Get all feature names.

```
168 {
169     return _featNameTable.getNames();
170 }
```

The documentation for this class was generated from the following files:

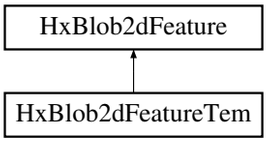
- **HxBlob2d.h**
- HxBlob2d.c

8.4 HxBlob2dFeature Class Reference

Base class for blob features.

`#include <HxBlob2dFeature.h>`

Inheritance diagram for HxBlob2dFeature::



Public Methods

- **HxBlob2dFeature** ()

  Constructor.

- virtual ∼**HxBlob2dFeature** ()

  Destructor.

- virtual HxBlob2dFeature ∗ **clone** () const=0

  Clone operation.

Protected Methods

- **HxBlob2dFeature** (const HxBlob2dFeature &)

  Copy constructor.

8.4.1 Detailed Description

Base class for blob features.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 HxBlob2dFeature::HxBlob2dFeature () [inline]

Constructor.

```
38 {
39 }
```

8.4.2.2 HxBlob2dFeature::∼HxBlob2dFeature () [inline, virtual]

Destructor.

```
43 {
44 }
```

8.4.2.3 HxBlob2dFeature::HxBlob2dFeature (const HxBlob2dFeature & *rhs*) [inline, protected]

Copy constructor.

```
48 {
49 }
```

8.4.3 Member Function Documentation

8.4.3.1 virtual HxBlob2dFeature* HxBlob2dFeature::clone () const [pure virtual]

Clone operation.

Reimplemented in **HxBlob2dFeatureTem** (p. 409).

The documentation for this class was generated from the following file:

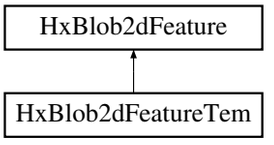
- **HxBlob2dFeature.h**

8.5 HxBlob2dFeatureTem Class Template Reference

Template specialization of **HxBlob2dFeature** (p. 406) to store any type as a feature.

`#include <HxBlob2dFeatureTem.h>`

Inheritance diagram for HxBlob2dFeatureTem::



Public Methods

- **HxBlob2dFeatureTem** (const ValT v)
  Constructor.
- virtual ~**HxBlob2dFeatureTem** ()
  Destructor.
- virtual **HxBlob2dFeature** * **clone** () const
  Clone operation.
- ValT **getValue** () const
  Get the value of this tag.

Protected Methods

- **HxBlob2dFeatureTem** (const HxBlob2dFeatureTem &rhs)
  Copy constructor.

8.5.1 Detailed Description

template<class ValT> class HxBlob2dFeatureTem< ValT >

Template specialization of **HxBlob2dFeature** (p. 406) to store any type as a feature.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 template<class ValT> HxBlob2dFeatureTem< ValT >::HxBlob2dFeatureTem (const ValT *v*) [inline]

Constructor.

```
46     : _value(v)
47 {
48 }
```

8.5.2.2 template<class ValT> HxBlob2dFeatureTem< ValT >::~HxBlob2dFeatureTem () [virtual]

Destructor.

```
59 {
60 }
```

8.5.2.3 template<class ValT> HxBlob2dFeatureTem< ValT >::HxBlob2dFeatureTem (const HxBlob2dFeatureTem< ValT > & *rhs*) [inline, protected]

Copy constructor.

```
53     : _value(rhs._value)
54 {
55 }
```

8.5.3 Member Function Documentation

8.5.3.1 template<class ValT> HxBlob2dFeature * HxBlob2dFeatureTem< ValT >::clone () const [virtual]

Clone operation.

Reimplemented from **HxBlob2dFeature** (p. 408).

```
65 {
66     return new HxBlob2dFeatureTem(*this);
67 }
```

8.5.3.2 template<class ValT> ValT HxBlob2dFeatureTem< ValT >::getValue () const [inline]

Get the value of this tag.

```
72 {
73     return _value;
74 }
```

The documentation for this class was generated from the following file:

- **HxBlob2dFeatureTem.h**

8.6 HxBlob2dList Class Reference

A list of HxBlob2dPtr's, that is pointers to **HxBlob2d** (p. 402)'s.

`#include <HxBlob2dList.h>`

Public Types

- typedef std::back_insert_iterator< HxBlob2dList > **back_insert_iterator**

  back inserter.

8.6.1 Detailed Description

A list of HxBlob2dPtr's, that is pointers to **HxBlob2d** (p. 402)'s.

8.6.2 Member Typedef Documentation

8.6.2.1 typedef std::back_insert_iterator<HxBlob2dList> HxBlob2dList::back_insert_iterator

back inserter.

The documentation for this class was generated from the following file:

- **HxBlob2dList.h**

8.7 HxBlob2dPtrLess Struct Reference

Class definition for ordering of **HxBlob2d** (p. 402)* based on getLabel for use in STL containers.

`#include <HxBlob2d.h>`

Public Methods

- bool **operator()** (**HxBlob2d** *b1, **HxBlob2d** *b2) const

8.7.1 Detailed Description

Class definition for ordering of **HxBlob2d** (p. 402)∗ based on getLabel for use in STL containers.

The documentation for this struct was generated from the following file:

- **HxBlob2d.h**

8.8 HxBlob2dRelation Class Reference

A relation in a set of blobs in 2D.

`#include <HxBlob2dRelation.h>`

Public Methods

- **HxBlob2dRelation** ()
  Constructor.
- ∼**HxBlob2dRelation** ()
  Destructor.
- int **ident** () const
  The identifier of this relation.
- void **add** (**HxBlob2d** ∗blob, **HxBlob2dList** rels)
  Add a blob and its relations.
- **HxBlob2dListConstIter getBlobBegin** () const
  Get begin of the blobs.
- **HxBlob2dListConstIter getBlobEnd** () const
  Get end (STL term) of the blobs.
- **HxBlob2dList findRelatedBlobs** (**HxBlob2d** ∗blob)
  Find relations of given blob.
- **HxBlob2dListConstIter findRelatedBlobsBegin** (**HxBlob2d** ∗blob) const
  Get begin of related blobs list.
- **HxBlob2dListConstIter findRelatedBlobsEnd** (**HxBlob2d** ∗blob) const
  Get end (STL term) of related blobs list.
- **HxBlob2dListBackInserter findRelatedBlobsInserter** (**HxBlob2d** ∗blob)
  Get a backinserter for related blobs list.
- STD_OSTREAM & **put** (STD_OSTREAM &os) const
  Put relation on stream.

8.8.1 Detailed Description

A relation in a set of blobs in 2D.

Basically, for each blob a list ”related” blobs is stored.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 HxBlob2dRelation::HxBlob2dRelation ()

Constructor.

```
16 {
17 }
```

8.8.2.2 HxBlob2dRelation::∼HxBlob2dRelation ()

Destructor.

```
20 {
21     // todo
22 }
```

8.8.3 Member Function Documentation

8.8.3.1 int HxBlob2dRelation::ident () const

The identifier of this relation.

8.8.3.2 void HxBlob2dRelation::add (HxBlob2d ∗ *blob*, HxBlob2dList *rels*)

Add a blob and its relations.

```
26 {
27     _head.push_back(blob);
28     _tail.push_back(rels);
29 }
```

8.8.3.3 HxBlob2dListConstIter HxBlob2dRelation::getBlobBegin () const

Get begin of the blobs.

```
33 {
34     return _head.begin();
35 }
```

8.8.3.4 HxBlob2dListConstIter HxBlob2dRelation::getBlobEnd () const

Get end (STL term) of the blobs.

```
39 {
40     return _head.end();
41 }
```

8.8.3.5 HxBlob2dList HxBlob2dRelation::findRelatedBlobs (HxBlob2d * *blob*)

Find relations of given blob.

```
45 {
46     for (int i=0 ; i<_head.size() ; i++)
47         if (_head[i] == blob)
48             return _tail[i];
49     return HxBlob2dList();
50 }
```

8.8.3.6 HxBlob2dListConstIter HxBlob2dRelation::findRelatedBlobsBegin (HxBlob2d * *blob*) const

Get begin of related blobs list.

```
54 {
55     for (int i=0 ; i<_head.size() ; i++)
56         if (_head[i] == blob)
57             return _tail[i].begin();
58     return HxBlob2dListConstIter();
59 }
```

8.8.3.7 HxBlob2dListConstIter HxBlob2dRelation::findRelatedBlobsEnd (HxBlob2d * *blob*) const

Get end (STL term) of related blobs list.

```
63 {
64     for (int i=0 ; i<_head.size() ; i++)
65         if (_head[i] == blob)
66             return _tail[i].end();
67     return HxBlob2dListConstIter();
68 }
```

8.8.3.8 HxBlob2dListBackInserter HxBlob2dRelation::findRelatedBlobsInserter (HxBlob2d * *blob*)

Get a backinserter for related blobs list.

```
72 {
73     for (int i=0 ; i<_head.size() ; i++)
74         if (_head[i] == blob)
75             return std::back_inserter(_tail[i]);
76     static HxBlob2dList dummyList;
77     return std::back_inserter(dummyList);
78 }
```

8.8.3.9 STD_OSTREAM & HxBlob2dRelation::put (STD_OSTREAM & *os*) const

Put relation on stream.

```
82 {
83     os << "Blob2dRelation " << STD_ENDL;
84     for(int i=0 ; i<_head.size() ; i++) {
85         os << "  blob " << _head[i]->ident() << " rels ";
86         for (HxBlob2dListConstIter b=_tail[i].begin() ;
87                                    b < _tail[i].end() ; b++)
88             os << (*b)->ident() << ", ";
89         os << STD_ENDL;
90     }
91     return os;
92 }
```

The documentation for this class was generated from the following files:

- **HxBlob2dRelation.h**
- HxBlob2dRelation.c

8.9 HxBoundingBox Class Reference

Definition of a bounding box.

`#include <HxBoundingBox.h>`

Public Methods

- **HxBoundingBox** (**HxPointZ** b, **HxPointZ** e)
  Constructor, begin and end point are included.
- **HxBoundingBox** (**HxSizes** s)
  Constructor.
- **HxPointZ begin** () const
  Begin point.
- **HxPointZ end** () const
  End point.
- **HxSizes size** () const
  Sizes.
- bool **isEmpty** () const
  Check validity.
- HxBoundingBox **unite** (const HxBoundingBox &arg) const
  Union of this and arg.
- HxBoundingBox **intersect** (const HxBoundingBox &arg) const

Intersection of this and arg.

- HxBoundingBox **extend** (**HxPointZ** p) const
  Extension of the box.
- HxBoundingBox **translate** (**HxPointZ** p) const
  Move the box.
- bool **includes** (**HxPointZ** p) const
  Check whether p is in the box.
- std::ostream & **put** (std::ostream &) const
  Put the box on the given stream.

8.9.1 Detailed Description

Definition of a bounding box.

Begin and end point are included in the box.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 HxBoundingBox::HxBoundingBox (HxPointZ *b*, HxPointZ *e*)

Constructor, begin and end point are included.

```
14     : _isEmpty(false)
15 {
16     _begin = b.inf(e);
17     _end = b.sup(e);
18 }
```

8.9.2.2 HxBoundingBox::HxBoundingBox (HxSizes *s*)

Constructor.

```
21     : _isEmpty(false)
22 {
23     if (s.inf(HxSizes(1, 1, 1)) != HxSizes(1, 1, 1))
24     {
25         _isEmpty = true;
26         s = HxSizes(0, 0, 0);
27     }
28     _begin = HxPointZ(0, 0, 0);
29     _end = HxPointZ(s.x()-1, s.y()-1, s.z()-1);
30 }
```

8.9.3 Member Function Documentation

8.9.3.1 HxPointZ HxBoundingBox::begin () const [inline]

Begin point.

```
68 {
69     return _begin;
70 }
```

8.9.3.2 HxPointZ HxBoundingBox::end () const [inline]

End point.

```
74 {
75     return _end;
76 }
```

8.9.3.3 HxSizes HxBoundingBox::size () const

Sizes.

```
34 {
35     HxPointZ s = _end - _begin + HxPointZ(1, 1, 1);
36     return HxSizes(s.x(), s.y(), s.z());
37 }
```

8.9.3.4 bool HxBoundingBox::isEmpty () const [inline]

Check validity.

```
80 {
81     return _isEmpty;
82 }
```

8.9.3.5 HxBoundingBox HxBoundingBox::unite (const HxBoundingBox & *arg*) const

Union of this and arg.

```
41 {
42     return HxBoundingBox(_begin.inf(arg._begin), _end.sup(arg._end));
43 }
```

8.9.3.6 HxBoundingBox HxBoundingBox::intersect (const HxBoundingBox & *arg*) const

Intersection of this and arg.

```
47 {
48     HxPointZ b = _begin.sup(arg._begin);
49     HxPointZ e = _end.inf(arg._end);
50     if (b.inf(e) == b)
51         return HxBoundingBox(b, e);
52     return HxBoundingBox(HxSizes(0, 0, 0));
53 }
```

8.9.3.7 HxBoundingBox HxBoundingBox::extend (HxPointZ *p*) const

Extension of the box.

```
57 {
58     return HxBoundingBox(_begin.inf(p), _end.sup(p));
59 }
```

8.9.3.8 HxBoundingBox HxBoundingBox::translate (HxPointZ *p*) const

Move the box.

```
63 {
64     return HxBoundingBox(_begin + p, _end + p);
65 }
```

8.9.3.9 bool HxBoundingBox::includes (HxPointZ *p*) const

Check whether p is in the box.

```
69 {
70     return (    (p.x() > _begin.x()) &&
71                 (p.y() > _begin.y()) &&
72                 (p.z() > _begin.z()) &&
73                 (p.x() < _end.x()) &&
74                 (p.y() < _end.y()) &&
75                 (p.z() < _end.z()) ) ? true : false;
76 }
```

8.9.3.10 std::ostream & HxBoundingBox::put (std::ostream & *os*) const

Put the box on the given stream.

```
80 {
81     if (_isEmpty)
82     {
83         return os << "empty";
84     }
85     else
86     {
87         return os << "{ " << _begin << ", " << _end << " }";
88     }
89 }
```

The documentation for this class was generated from the following files:

- **HxBoundingBox.h**
- HxBoundingBox.c

8.10 HxBpoAdd Class Template Reference

Pixel functor for computation of addition.

`#include <HxBpoAdd.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.10.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoAdd< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of addition.

8.10.2 Member Typedef Documentation

8.10.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoAdd::TransVarianceCategory

Functor is translation invariant.

8.10.3 Constructor & Destructor Documentation

8.10.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoAdd< DstValT, Src1ValT, Src2ValT >::HxBpoAdd (HxTagList &) [inline]

Constructor : empty.

```
30                          {}
```

8.10.4 Member Function Documentation

8.10.4.1 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 #.

```
34                          { return x + y; }
```

8.10.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoAdd< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "add".

```
41                          { return HxString("add"); }
```

The documentation for this class was generated from the following file:

- **HxBpoAdd.h**

8.11 HxBpoAddAssign Struct Template Reference

Pixel functor for computation of addition assignment.

`#include <HxBpoAddAssign.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.11.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoAddAssign< DstValT, SrcValT >

Pixel functor for computation of addition assignment.

8.11.2 Member Typedef Documentation

8.11.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoAddAssign::TransVarianceCategory

Functor is translation invariant.

8.11.3 Constructor & Destructor Documentation

8.11.3.1 template<class DstValT, class SrcValT> HxBpoAddAssign< DstValT, SrcValT >::HxBpoAddAssign (HxTagList &) [inline]

Constructor : empty.

```
32                          {}
```

8.11.4 Member Function Documentation

8.11.4.1 template<class DstValT, class SrcValT> void HxBpoAddAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x += y #.

```
36                          { x += y; }
```

8.11.4.2 template<class DstValT, class SrcValT> HxString HxBpoAddAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”addAssign”.

```
43                          { return HxString("addAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoAddAssign.h**

8.12 HxBpoAddSat Class Template Reference

Pixel functor for computation of saturated addition.

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxBpoAddSat** (**HxTagList** &t)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &arg1, const Src2ValT &arg2)
  Actual operation : # if(x+y)<=MAXVAL return (x + y) else return MAXVAL #.

Static Public Methods

- **HxString className** ()
  The name : "addSat".

8.12.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoAddSat< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of saturated addition.

8.12.2 Member Typedef Documentation

8.12.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoAddSat::TransVarianceCategory

Functor is translation invariant.

8.12.3 Constructor & Destructor Documentation

8.12.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoAddSat< DstValT, Src1ValT, Src2ValT >::HxBpoAddSat (HxTagList & *t*) [inline]

Constructor : empty.

```
31                      {
32 //better use taglist to get minSat maxSat values
33                          int maxSat = HxGetTag<int>(t, "maxSat", 0);
34                          for (int i=1; i<=maxValSat.dim(); i++)
35                              maxValSat.setValue(i, maxSat);
36 //lt                        maxValSat.setValue(i, std::numeric_limits<Src1ValT::ValueType>::max());
37 //                        std::cout << "welcome in " << typeid(*this).name() << ", maxValSat=" << maxValSat << std::end
38
39                      }
```

8.12.4 Member Function Documentation

8.12.4.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoAddSat< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *arg1*, const Src2ValT & *arg2*) [inline]

Actual operation : # if(x+y)<=MAXVAL return (x + y) else return MAXVAL #.

```
47                          {
48                              DstValT ret;
49                              for (int i=1; i<=arg1.dim(); i++) {
50                                  ret.setValue(i,
51                                      (double)arg1.getValue(i)+(double)arg2.getValue(i)>maxValSat.getValue(i) ?
52                                      maxValSat.getValue(i) :
53                                      arg1.getValue(i)+arg2.getValue(i)
54                                  );
55                              }
56                              return ret;
57                          }
```

8.12.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoAddSat< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "addSat".

```
61                          { return HxString("addSat"); }
```

The documentation for this class was generated from the following file:

- HxAddSat.c

8.13 HxBpoAnd Class Template Reference

Pixel functor for computation of and.

`#include <HxBpoAnd.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.13.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoAnd< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of and.

8.13.2 Member Typedef Documentation

8.13.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoAnd::TransVarianceCategory

Functor is translation invariant.

8.13.3 Constructor & Destructor Documentation

8.13.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoAnd< DstValT, Src1ValT, Src2ValT >::HxBpoAnd (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.13.4 Member Function Documentation

8.13.4.1 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) #.

```
33                          { return x.and(y); }
```

8.13.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoAnd< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "and".

```
40                          { return HxString("and"); }
```

The documentation for this class was generated from the following file:

- **HxBpoAnd.h**

8.14 HxBpoBind2Val Class Template Reference

Pixel functor for binding of second value of a binary pixel operation.

`#include <HxBpoBind2Val.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxBpoBind2Val** (**HxTagList** &tags)
  Constructor : obtain value from taglist.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : bpoOp (x, value).

Static Public Methods

- **HxString className** ()
  The name is that of BpoOpT.

8.14.1 Detailed Description

template<class DstValT, class SrcValT, class BpoOpT> class HxBpoBind2Val< DstValT, SrcValT, BpoOpT >

Pixel functor for binding of second value of a binary pixel operation.

In other words, this class turns the binary operation BpoOpT into a unary operation by using a constant value obtained from the taglist.

8.14.2 Member Typedef Documentation

8.14.2.1 template<class DstValT, class SrcValT, class BpoOpT> typedef HxTagTransInVar HxBpoBind2Val::TransVarianceCategory

Functor is translation invariant.

8.14.3 Constructor & Destructor Documentation

8.14.3.1 template<class DstValT, class SrcValT, class BpoOpT> HxBpoBind2Val< DstValT, SrcValT, BpoOpT >::HxBpoBind2Val (HxTagList & *tags*) [inline]

Constructor : obtain value from taglist.

```
29                                                  : _bpoOp(tags)
30                          { _value = HxGetTag<HxValue>(tags, "value"); }
```

8.14.4 Member Function Documentation

8.14.4.1 template<class DstValT, class SrcValT, class BpoOpT> DstValT HxBpoBind2Val< DstValT, SrcValT, BpoOpT >::doIt (const SrcValT & *x*) [inline]

Actual operation : bpoOp (x, value).

```
34                          { return _bpoOp.doIt(x, _value); }
```

8.14.4.2 template<class DstValT, class SrcValT, class BpoOpT> HxString HxBpoBind2Val< DstValT, SrcValT, BpoOpT >::className () [inline, static]

The name is that of BpoOpT.

```
38                          { return BpoOpT::className(); }
```

The documentation for this class was generated from the following file:

- **HxBpoBind2Val.h**

8.15 HxBpoCross Class Template Reference

Pixel functor for computation of cross product.

`#include <HxBpoCross.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.15.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoCross< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of cross product.

8.15.2 Member Typedef Documentation

8.15.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoCross::TransVarianceCategory

Functor is translation invariant.

8.15.3 Constructor & Destructor Documentation

8.15.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoCross< DstValT, Src1ValT, Src2ValT >::HxBpoCross (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.15.4 Member Function Documentation

8.15.4.1 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) #.

```
33                          { return x.cross(y); }
```

8.15.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoCross< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”cross”.

```
37                          { return HxString("cross"); }
```

The documentation for this class was generated from the following file:

- **HxBpoCross.h**

8.16 HxBpoDiv Class Template Reference

Pixel functor for computation of division.

```
#include <HxBpoDiv.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.16.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoDiv< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of division.

8.16.2 Member Typedef Documentation

8.16.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoDiv::TransVarianceCategory

Functor is translation invariant.

8.16.3 Constructor & Destructor Documentation

8.16.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoDiv< DstValT, Src1ValT, Src2ValT >::HxBpoDiv (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.16.4 Member Function Documentation

8.16.4.1 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 #.

```
33                          { return x / y; }
```

8.16.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoDiv< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "div".

```
40                          { return HxString("div"); }
```

The documentation for this class was generated from the following file:

- **HxBpoDiv.h**

8.17 HxBpoDot Class Template Reference

Pixel functor for computation of dot product.

`#include <HxBpoDot.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.17.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoDot< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of dot product.

8.17.2 Member Typedef Documentation

8.17.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoDot::TransVarianceCategory

Functor is translation invariant.

8.17.3 Constructor & Destructor Documentation

8.17.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoDot< DstValT, Src1ValT, Src2ValT >::HxBpoDot (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.17.4 Member Function Documentation

8.17.4.1 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) #.

```
33                          { return x.dot(y); }
```

8.17.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoDot< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”dot”.

```
37                          { return HxString("dot"); }
```

The documentation for this class was generated from the following file:

- **HxBpoDot.h**

8.18 HxBpoEqual Class Template Reference

Pixel functor for computation of equal.

`#include <HxBpoEqual.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.18.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of equal.

8.18.2 Member Typedef Documentation

8.18.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoEqual::TransVarianceCategory

Functor is translation invariant.

8.18.3 Constructor & Destructor Documentation

8.18.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoEqual< DstValT, Src1ValT, Src2ValT >::HxBpoEqual (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.18.4 Member Function Documentation

8.18.4.1 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 #.

```
33                          { return x == y; }
```

8.18.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "equal".

```
37                          { return HxString("equal"); }
```

The documentation for this class was generated from the following file:

- **HxBpoEqual.h**

8.19 HxBpoGreaterEqual Class Template Reference

Pixel functor for computation of greater equal.

`#include <HxBpoGreaterEqual.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.19.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of greater equal.

8.19.2 Member Typedef Documentation

8.19.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoGreaterEqual::TransVarianceCategory

Functor is translation invariant.

8.19.3 Constructor & Destructor Documentation

8.19.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >::HxBpoGreaterEqual (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.19.4 Member Function Documentation

8.19.4.1 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 #.

```
33                          { return x >= y; }
```

8.19.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoGreaterEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”greaterEqual”.

```
37                          { return HxString("greaterEqual"); }
```

The documentation for this class was generated from the following file:

- **HxBpoGreaterEqual.h**

8.20 HxBpoGreaterThan Class Template Reference

Pixel functor for computation of greater than.

```
#include <HxBpoGreaterThan.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.20.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of greater than.

8.20.2 Member Typedef Documentation

8.20.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoGreaterThan::TransVarianceCategory

Functor is translation invariant.

8.20.3 Constructor & Destructor Documentation

8.20.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >::HxBpoGreaterThan (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.20.4 Member Function Documentation

8.20.4.1 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 #.

```
33                          { return x > y; }
```

8.20.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoGreaterThan< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "greaterThan".

```
37                          { return HxString("greaterThan"); }
```

The documentation for this class was generated from the following file:

- **HxBpoGreaterThan.h**

8.21 HxBpoHighlightRegion Class Template Reference

Pixel functor for region highlighting.

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxBpoHighlightRegion** (**HxTagList** &tags)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x, const LabelValT &y)
  Actual operation.

Static Public Methods

- DstValT **neutralElement** ()
- **HxString className** ()
  The name : "highlightRegion".

8.21.1 Detailed Description

template<class DstValT, class SrcValT, class LabelValT> class HxBpoHighlightRegion< DstValT, SrcValT, LabelValT >

Pixel functor for region highlighting.

8.21.2 Member Typedef Documentation

8.21.2.1 template<class DstValT, class SrcValT, class LabelValT> typedef HxTagTransInVar HxBpoHighlightRegion::TransVarianceCategory

Functor is translation invariant.

8.21.3 Constructor & Destructor Documentation

8.21.3.1 template<class DstValT, class SrcValT, class LabelValT> HxBpoHighlightRegion< DstValT, SrcValT, LabelValT >::HxBpoHighlightRegion (HxTagList & *tags*) [inline]

Constructor : empty.

```
30                          {
31                              _label = LabelValT(HxGetTag<int>(tags, "label", 1));
32                              _factor = HxScalarDouble(
33                                  HxGetTag<double>(tags,"factor", 0.75));
34                          }
```

8.21.4 Member Function Documentation

8.21.4.1 template<class DstValT, class SrcValT, class LabelValT> DstValT HxBpoHighlightRegion< DstValT, SrcValT, LabelValT >::doIt (const SrcValT & *x*, const LabelValT & *y*) [inline]

Actual operation.

```
38                          { return (y == _label) ? x : x*_factor; }
```

8.21.4.2 template<class DstValT, class SrcValT, class LabelValT> HxString HxBpoHighlightRegion< DstValT, SrcValT, LabelValT >::className () [inline, static]

The name : ”highlightRegion”.

```
45                          { return HxString("highlightRegion"); }
```

The documentation for this class was generated from the following file:

- HxHighlightRegion.c

8.22 HxBpoInf Class Template Reference

Pixel functor for computation of infimum.

`#include <HxBpoInf.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.22.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoInf< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of infimum.

8.22.2 Member Typedef Documentation

8.22.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoInf::TransVarianceCategory

Functor is translation invariant.

8.22.3 Constructor & Destructor Documentation

8.22.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoInf< DstValT, Src1ValT, Src2ValT >::HxBpoInf (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.22.4 Member Function Documentation

8.22.4.1 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) #.

```
33                          { return x.inf(y); }
```

8.22.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoInf< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "inf".

```
40                          { return HxString("inf"); }
```

The documentation for this class was generated from the following file:

- **HxBpoInf.h**

8.23 HxBpoInfAssign Struct Template Reference

Pixel functor for computation of infimum assignment.

`#include <HxBpoInfAssign.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.23.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoInfAssign< DstValT, SrcValT >

Pixel functor for computation of infimum assignment.

8.23.2 Member Typedef Documentation

8.23.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoInfAssign::TransVarianceCategory

Functor is translation invariant.

8.23.3 Constructor & Destructor Documentation

8.23.3.1 template<class DstValT, class SrcValT> HxBpoInfAssign< DstValT, SrcValT >::HxBpoInfAssign (HxTagList &) [inline]

Constructor : empty.

```
30                          {}
```

8.23.4 Member Function Documentation

8.23.4.1 template<class DstValT, class SrcValT> void HxBpoInfAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.infAssign(y) #.

```
34                          { x.infAssign(y); }
```

8.23.4.2 template<class DstValT, class SrcValT> HxString HxBpoInfAssign< DstValT, SrcValT >::className () [inline, static]

The name : "infAssign".

```
41                          { return HxString("infAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoInfAssign.h**

8.24 HxBpoLeftShift Class Template Reference

Pixel functor for computation of left shift.

`#include <HxBpoLeftShift.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.24.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of left shift.

8.24.2 Member Typedef Documentation

8.24.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoLeftShift::TransVarianceCategory

Functor is translation invariant.

8.24.3 Constructor & Destructor Documentation

8.24.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >::HxBpoLeftShift (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.24.4 Member Function Documentation

8.24.4.1 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) #.

```
33                          { return x.leftShift(y); }
```

8.24.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoLeftShift< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”leftShift”.

```
40                          { return HxString("leftShift"); }
```

The documentation for this class was generated from the following file:

- **HxBpoLeftShift.h**

8.25 HxBpoLessEqual Class Template Reference

Pixel functor for computation of less equal.

`#include <HxBpoLessEqual.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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”.

8.25.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of less equal.

8.25.2 Member Typedef Documentation

8.25.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoLessEqual::TransVarianceCategory

Functor is translation invariant.

8.25.3 Constructor & Destructor Documentation

8.25.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >::HxBpoLessEqual (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.25.4 Member Function Documentation

8.25.4.1 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 #.

```
33                          { return x <= y; }
```

8.25.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoLessEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”lessEqual”.

```
37                          { return HxString("lessEqual"); }
```

The documentation for this class was generated from the following file:

- **HxBpoLessEqual.h**

8.26 HxBpoLessThan Class Template Reference

Pixel functor for computation of less than.

`#include <HxBpoLessThan.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.26.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoLessThan< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of less than.

8.26.2 Member Typedef Documentation

8.26.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoLessThan::TransVarianceCategory

Functor is translation invariant.

8.26.3 Constructor & Destructor Documentation

8.26.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoLessThan< DstValT, Src1ValT, Src2ValT >::HxBpoLessThan (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.26.4 Member Function Documentation

8.26.4.1 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 #.

```
33                          { return x < y; }
```

8.26.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoLessThan< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "lessThan".

```
37                          { return HxString("lessThan"); }
```

The documentation for this class was generated from the following file:

- **HxBpoLessThan.h**

8.27 HxBpoMax Class Template Reference

Pixel functor for computation of maximum.

`#include <HxBpoMax.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.27.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMax< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of maximum.

8.27.2 Member Typedef Documentation

8.27.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoMax::TransVarianceCategory

Functor is translation invariant.

8.27.3 Constructor & Destructor Documentation

8.27.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMax< DstValT, Src1ValT, Src2ValT >::HxBpoMax (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.27.4 Member Function Documentation

8.27.4.1 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) #.

```
33                          { return x.max(y); }
```

8.27.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMax< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”max”.

```
40                          { return HxString("max"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMax.h**

8.28 HxBpoMaxAssign Struct Template Reference

Pixel functor for computation of maximum assignment.

```
#include <HxBpoMaxAssign.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.28.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoMaxAssign< DstValT, SrcValT >

Pixel functor for computation of maximum assignment.

8.28.2 Member Typedef Documentation

8.28.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoMaxAssign::TransVarianceCategory

Functor is translation invariant.

8.28.3 Constructor & Destructor Documentation

8.28.3.1 template<class DstValT, class SrcValT> HxBpoMaxAssign< DstValT, SrcValT >::HxBpoMaxAssign (HxTagList &) [inline]

Constructor : empty.

```
31                          {}
```

8.28.4 Member Function Documentation

8.28.4.1 template<class DstValT, class SrcValT> void HxBpoMaxAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.maxAssign(y) #.

```
35                          { x.maxAssign(y); }
```

8.28.4.2 template<class DstValT, class SrcValT> HxString HxBpoMaxAssign< DstValT, SrcValT >::className () [inline, static]

The name : "maxAssign".

```
42                          { return HxString("maxAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoMaxAssign.h**

8.29 HxBpoMin Class Template Reference

Pixel functor for computation of minimum.

`#include <HxBpoMin.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.29.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMin< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of minimum.

8.29.2 Member Typedef Documentation

8.29.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoMin::TransVarianceCategory

Functor is translation invariant.

8.29.3 Constructor & Destructor Documentation

8.29.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMin< DstValT, Src1ValT, Src2ValT >::HxBpoMin (HxTagList &) [inline]

Constructor : empty.

```
30                          {}
```

8.29.4 Member Function Documentation

8.29.4.1 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) #.

```
34                          { return x.min(y); }
```

8.29.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMin< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”min”.

```
41                          { return HxString("min"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMin.h**

8.30 HxBpoMinAssign Struct Template Reference

Pixel functor for computation of minimum assignment.

`#include <HxBpoMinAssign.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.30.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoMinAssign< DstValT, SrcValT >

Pixel functor for computation of minimum assignment.

8.30.2 Member Typedef Documentation

8.30.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoMinAssign::TransVarianceCategory

Functor is translation invariant.

8.30.3 Constructor & Destructor Documentation

8.30.3.1 template<class DstValT, class SrcValT> HxBpoMinAssign< DstValT, SrcValT >::HxBpoMinAssign (HxTagList &) [inline]

Constructor : empty.

```
31                          {}
```

8.30.4 Member Function Documentation

8.30.4.1 template<class DstValT, class SrcValT> void HxBpoMinAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.minAssign(y) #.

```
35                          { x.minAssign(y); }
```

8.30.4.2 template<class DstValT, class SrcValT> HxString HxBpoMinAssign< DstValT, SrcValT >::className () [inline, static]

The name : "minAssign".

```
42                          { return HxString("minAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoMinAssign.h**

8.31 HxBpoMod Class Template Reference

Pixel functor for computation of modulo.

`#include <HxBpoMod.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.31.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMod< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of modulo.

8.31.2 Member Typedef Documentation

8.31.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoMod::TransVarianceCategory

Functor is translation invariant.

8.31.3 Constructor & Destructor Documentation

8.31.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMod< DstValT, Src1ValT, Src2ValT >::HxBpoMod (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.31.4 Member Function Documentation

8.31.4.1 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) #.

```
33                          { return x.mod(y); }
```

8.31.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMod< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”mod”.

```
37                          { return HxString("mod"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMod.h**

8.32 HxBpoMul Class Template Reference

Pixel functor for computation of multiplication.

```
#include <HxBpoMul.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.32.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoMul< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of multiplication.

8.32.2 Member Typedef Documentation

8.32.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoMul::TransVarianceCategory

Functor is translation invariant.

8.32.3 Constructor & Destructor Documentation

8.32.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoMul< DstValT, Src1ValT, Src2ValT >::HxBpoMul (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.32.4 Member Function Documentation

8.32.4.1 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 #.

```
33                          { return x * y; }
```

8.32.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoMul< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "mul".

```
40                          { return HxString("mul"); }
```

The documentation for this class was generated from the following file:

- **HxBpoMul.h**

8.33 HxBpoMulAssign Struct Template Reference

Pixel functor for computation of multiplication assignment.

`#include <HxBpoMulAssign.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.33.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoMulAssign< DstValT, SrcValT >

Pixel functor for computation of multiplication assignment.

8.33.2 Member Typedef Documentation

8.33.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoMulAssign::TransVarianceCategory

Functor is translation invariant.

8.33.3 Constructor & Destructor Documentation

8.33.3.1 template<class DstValT, class SrcValT> HxBpoMulAssign< DstValT, SrcValT >::HxBpoMulAssign (HxTagList &) [inline]

Constructor : empty.

```
32                          {}
```

8.33.4 Member Function Documentation

8.33.4.1 template<class DstValT, class SrcValT> void HxBpoMulAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x *= y #.

```
36                          { x *= y; }
```

8.33.4.2 template<class DstValT, class SrcValT> HxString HxBpoMulAssign< DstValT, SrcValT >::className () [inline, static]

The name : "mulAssign".

```
43                          { return HxString("mulAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoMulAssign.h**

8.34 HxBpoNotEqual Class Template Reference

Pixel functor for computation of not equal.

`#include <HxBpoNotEqual.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.34.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of not equal.

8.34.2 Member Typedef Documentation

8.34.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoNotEqual::TransVarianceCategory

Functor is translation invariant.

8.34.3 Constructor & Destructor Documentation

8.34.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >::HxBpoNotEqual (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.34.4 Member Function Documentation

8.34.4.1 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 #.

```
33                          { return x != y; }
```

8.34.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoNotEqual< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "notEqual".

```
37                          { return HxString("notEqual"); }
```

The documentation for this class was generated from the following file:

- **HxBpoNotEqual.h**

8.35 HxBpoOr Class Template Reference

Pixel functor for computation of or.

`#include <HxBpoOr.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.35.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoOr< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of or.

8.35.2 Member Typedef Documentation

8.35.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoOr::TransVarianceCategory

Functor is translation invariant.

8.35.3 Constructor & Destructor Documentation

8.35.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoOr< DstValT, Src1ValT, Src2ValT >::HxBpoOr (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.35.4 Member Function Documentation

8.35.4.1 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) #.

```
33                          { return x.or(y); }
```

8.35.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoOr< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”or”.

```
40                          { return HxString("or"); }
```

The documentation for this class was generated from the following file:

- **HxBpoOr.h**

8.36 HxBpoPow Class Template Reference

Pixel functor for computation of power.

```
#include <HxBpoPow.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.36.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoPow< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of power.

8.36.2 Member Typedef Documentation

8.36.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoPow::TransVarianceCategory

Functor is translation invariant.

8.36.3 Constructor & Destructor Documentation

8.36.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoPow< DstValT, Src1ValT, Src2ValT >::HxBpoPow (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.36.4 Member Function Documentation

8.36.4.1 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) #.

```
33                          { return x.pow(y); }
```

8.36.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoPow< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "pow".

```
37                          { return HxString("pow"); }
```

The documentation for this class was generated from the following file:

- **HxBpoPow.h**

8.37 HxBpoRightShift Class Template Reference

Pixel functor for computation of right shift.

`#include <HxBpoRightShift.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.37.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoRightShift< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of right shift.

8.37.2 Member Typedef Documentation

8.37.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoRightShift::TransVarianceCategory

Functor is translation invariant.

8.37.3 Constructor & Destructor Documentation

8.37.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoRightShift< DstValT, Src1ValT, Src2ValT >::HxBpoRightShift (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.37.4 Member Function Documentation

8.37.4.1 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) #.

```
33                          { return x.rightShift(y); }
```

8.37.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoRightShift< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”rightShift”.

```
40                          { return HxString("rightShift"); }
```

The documentation for this class was generated from the following file:

- **HxBpoRightShift.h**

8.38 HxBpoSqrDst Class Template Reference

Pixel functor for computation of squared distance.

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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".

8.38.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of squared distance.

8.38.2 Member Typedef Documentation

8.38.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoSqrDst::TransVarianceCategory

Functor is translation invariant.

8.38.3 Constructor & Destructor Documentation

8.38.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >::HxBpoSqrDst (HxTagList &) [inline]

Constructor : empty.

```
29                          { }
```

8.38.4 Member Function Documentation

8.38.4.1 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 #.

```
33                          { return (arg1 - arg2) * (arg1 - arg2); }
```

8.38.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSqrDst< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : "sqrDst".

```
37                          { return HxString("sqrDst"); }
```

The documentation for this class was generated from the following file:

- HxSquaredDistance.c

8.39 HxBpoSub Class Template Reference

Pixel functor for computation of subtraction.

`#include <HxBpoSub.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.39.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSub< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of subtraction.

8.39.2 Member Typedef Documentation

8.39.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoSub::TransVarianceCategory

Functor is translation invariant.

8.39.3 Constructor & Destructor Documentation

8.39.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSub< DstValT, Src1ValT, Src2ValT >::HxBpoSub (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.39.4 Member Function Documentation

8.39.4.1 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 #.

```
33                          { return x - y; }
```

8.39.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSub< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”sub”.

```
40                          { return HxString("sub"); }
```

The documentation for this class was generated from the following file:

- **HxBpoSub.h**

8.40 HxBpoSubAssign Struct Template Reference

Pixel functor for computation of subtraction assignment.

```
#include <HxBpoSubAssign.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.40.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoSubAssign< DstValT, SrcValT >

Pixel functor for computation of subtraction assignment.

8.40.2 Member Typedef Documentation

8.40.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoSubAssign::TransVarianceCategory

Functor is translation invariant.

8.40.3 Constructor & Destructor Documentation

8.40.3.1 template<class DstValT, class SrcValT> HxBpoSubAssign< DstValT, SrcValT >::HxBpoSubAssign (HxTagList &) [inline]

Constructor : empty.

```
32                          {}
```

8.40.4 Member Function Documentation

8.40.4.1 template<class DstValT, class SrcValT> void HxBpoSubAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x -= y #.

```
36                          { x -= y; }
```

8.40.4.2 template<class DstValT, class SrcValT> HxString HxBpoSubAssign< DstValT, SrcValT >::className () [inline, static]

The name : "subAssign".

```
43                          { return HxString("subAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoSubAssign.h**

8.41 HxBpoSubSat Class Template Reference

Pixel functor for computation of saturated addition.

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxBpoSubSat** (**HxTagList** &t)
  Constructor : empty.
- DstValT **doIt** (const Src1ValT &arg1, const Src2ValT &arg2)
  Actual operation : # if(x+y)<=MAXVAL return (x + y) else return MAXVAL #.

Static Public Methods

- **HxString className** ()
  The name : "SubSat".

8.41.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSubSat< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of saturated addition.

8.41.2 Member Typedef Documentation

8.41.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoSubSat::TransVarianceCategory

Functor is translation invariant.

8.41.3 Constructor & Destructor Documentation

8.41.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSubSat< DstValT, Src1ValT, Src2ValT >::HxBpoSubSat (HxTagList & *t*) [inline]

Constructor : empty.

```
31                      {
32 //better use taglist to get minSat maxSat values
33                          int minSat = HxGetTag<int>(t, "minSat", 0);
34                          for (int i=1; i<=minValSat.dim(); i++)
35                              minValSat.setValue(i, minSat);
36 //lt                        maxValSat.setValue(i, std::numeric_limits<Src1ValT::ValueType>::max());
37 //                        std::cout << "welcome in " << typeid(*this).name() << ", maxValSat=" << maxValSat << std::end
38
39                      }
```

8.41.4 Member Function Documentation

8.41.4.1 template<class DstValT, class Src1ValT, class Src2ValT> DstValT HxBpoSubSat< DstValT, Src1ValT, Src2ValT >::doIt (const Src1ValT & *arg1*, const Src2ValT & *arg2*) [inline]

Actual operation : # if(x+y)<=MAXVAL return (x + y) else return MAXVAL #.

```
47                          {
48                              DstValT ret;
49                              for (int i=1; i<=arg1.dim(); i++) {
50                                  ret.setValue(i,
51                                      (double)arg1.getValue(i)-(double)arg2.getValue(i)<minValSat.getValue(i) ?
52                                      minValSat.getValue(i) :
53                                      arg1.getValue(i)-arg2.getValue(i)
54                                  );
55                              }
56                              return ret;
57                          }
```

8.41.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSubSat< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”SubSat”.

```
61                          { return HxString("subSat"); }
```

The documentation for this class was generated from the following file:

- HxSubSat.c

8.42 HxBpoSup Class Template Reference

Pixel functor for computation of supremum.

`#include <HxBpoSup.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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”.

8.42.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoSup< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of supremum.

8.42.2 Member Typedef Documentation

8.42.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoSup::TransVarianceCategory

Functor is translation invariant.

8.42.3 Constructor & Destructor Documentation

8.42.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoSup< DstValT, Src1ValT, Src2ValT >::HxBpoSup (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.42.4 Member Function Documentation

8.42.4.1 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) #.

```
33                          { return x.sup(y); }
```

8.42.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoSup< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”sup”.

```
40                          { return HxString("sup"); }
```

The documentation for this class was generated from the following file:

- **HxBpoSup.h**

8.43 HxBpoSupAssign Struct Template Reference

Pixel functor for computation of supremum assignment.

`#include <HxBpoSupAssign.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- 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".

8.43.1 Detailed Description

template<class DstValT, class SrcValT> struct HxBpoSupAssign< DstValT, SrcValT >

Pixel functor for computation of supremum assignment.

8.43.2 Member Typedef Documentation

8.43.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxBpoSupAssign::TransVarianceCategory

Functor is translation invariant.

8.43.3 Constructor & Destructor Documentation

8.43.3.1 template<class DstValT, class SrcValT> HxBpoSupAssign< DstValT, SrcValT >::HxBpoSupAssign (HxTagList &) [inline]

Constructor : empty.

```
31                          {}
```

8.43.4 Member Function Documentation

8.43.4.1 template<class DstValT, class SrcValT> void HxBpoSupAssign< DstValT, SrcValT >::doIt (DstValT & *x*, const SrcValT & *y*) [inline]

Actual operation : # x.supAssign(y) #.

```
35                          { x.supAssign(y); }
```

8.43.4.2 template<class DstValT, class SrcValT> HxString HxBpoSupAssign< DstValT, SrcValT >::className () [inline, static]

The name : ”supAssign”.

```
42                          { return HxString("supAssign"); }
```

The documentation for this struct was generated from the following file:

- **HxBpoSupAssign.h**

8.44 HxBpoXor Class Template Reference

Pixel functor for computation of exclusive or.

`#include <HxBpoXor.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

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”.

8.44.1 Detailed Description

template<class DstValT, class Src1ValT, class Src2ValT> class HxBpoXor< DstValT, Src1ValT, Src2ValT >

Pixel functor for computation of exclusive or.

8.44.2 Member Typedef Documentation

8.44.2.1 template<class DstValT, class Src1ValT, class Src2ValT> typedef HxTagTransInVar HxBpoXor::TransVarianceCategory

Functor is translation invariant.

8.44.3 Constructor & Destructor Documentation

8.44.3.1 template<class DstValT, class Src1ValT, class Src2ValT> HxBpoXor< DstValT, Src1ValT, Src2ValT >::HxBpoXor (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.44.4 Member Function Documentation

8.44.4.1 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) #.

```
33                          { return x.xor(y); }
```

8.44.4.2 template<class DstValT, class Src1ValT, class Src2ValT> HxString HxBpoXor< DstValT, Src1ValT, Src2ValT >::className () [inline, static]

The name : ”xor”.

```
37                          { return HxString("xor"); }
```

The documentation for this class was generated from the following file:

- **HxBpoXor.h**

8.45 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**

8.45.1 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.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 HxBSplineBasis::HxBSplineBasis ()

Construct default basis.

```
17 {
18     makeDefault();
19 }
```

8.45.2.2 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 }
```

8.45.2.3 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.

8.45.2.4 HxBSplineBasis::~HxBSplineBasis ()

Destructor.

```
89 {
90 }
```

8.45.3 Member Function Documentation

8.45.3.1 HxBSplineType HxBSplineBasis::curveType () const [inline]

Get the type of the cuve.

```
156 {
157     return _curveType;
158 }
```

8.45.3.2 int HxBSplineBasis::degree () const [inline]

Get the degree.

```
162 {
163     return _degree;
164 }
```

8.45.3.3 int HxBSplineBasis::nIntervals () const [inline]

Get the number of intervals.

```
169 {
170     return _nIntervals;
171 }
```

8.45.3.4 HxBSplineKnotsAlg HxBSplineBasis::knotsType () const [inline]

Get the type of the knots generating algorithm.

```
175 {
176     return _knotsType;
177 }
```

8.45.3.5 double HxBSplineBasis::minT () const [inline]

Get the minimum value for t.

```
181 {
182     return _minT;
183 }
```

8.45.3.6 double HxBSplineBasis::maxT () const [inline]

Get the maximum value for t.

```
187 {
188     return _maxT;
189 }
```

8.45.3.7 double HxBSplineBasis::knot (int *i*) const [inline]

Get the value of the given knot.

```
208 {
209     return _knotsVec[i];
210 }
```

8.45.3.8 vector< double > HxBSplineBasis::allKnots () const [inline]

Get values of all knots.

```
214 {
215     return _knotsVec;
216 }
```

8.45.3.9 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 }
```

8.45.3.10 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 }
```

8.45.3.11 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 }
```

8.45.3.12 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 }
```

8.45.3.13 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 }
```

8.45.3.14 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

8.46 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).

8.46.1 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.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 HxBSplineCurve::HxBSplineCurve ()

Construct default curve.

```
21 {
22     _ident = _nr++;
23     makeDefault();
24 }
```

8.46.2.2 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 }
```

8.46.2.3 HxBSplineCurve::∼HxBSplineCurve ()

Destructor.

```
64 {
65 }
```

8.46.3 Member Function Documentation

8.46.3.1 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 }
```

8.46.3.2 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 }
```

8.46.3.3 int HxBSplineCurve::ident () const [inline]

Get the identifier.

```
200 {
201     return _ident;
202 }
```

8.46.3.4 HxBSplineBasis HxBSplineCurve::basis () const [inline]

Get the basis.

```
206 {
207     return _basis;
208 }
```

8.46.3.5 HxBSplineType HxBSplineCurve::curveType () const [inline]

Get the curve type.

```
212 {
213     return _basis.curveType();
214 }
```

8.46.3.6 int HxBSplineCurve::degree () const [inline]

Get the degree of the curve.

```
218 {
219     return _basis.degree();
220 }
```

8.46.3.7 double HxBSplineCurve::minT () const [inline]

Get the minimum value of T.

```
224 {
225     return _basis.minT();
226 }
```

8.46.3.8 double HxBSplineCurve::maxT () const [inline]

Get the maximum value of T.

```
230 {
231     return _basis.maxT();
232 }
```

8.46.3.9 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 }
```

8.46.3.10 int HxBSplineCurve::numP () const [inline]

Get the number of control points.

```
261 {
262     return _basis.numB();
263 }
```

8.46.3.11 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 }
```

8.46.3.12 HxPointSetR2 HxBSplineCurve::allP () const [inline]

Get the coordinates of all control points.

```
277 {
278     return _PVec;
279 }
```

8.46.3.13 HxPolyline2d HxBSplineCurve::controlP () const [inline]

Get the control polygon.

```
283 {
284     return HxPolyline2d(_PVec, (curveType() == closed));
285 }
```

8.46.3.14 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 }
```

8.46.3.15 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 }
```

8.46.3.16 HxBSplineInterval HxBSplineCurve::pathAffectedBy (int *index*) const [inline]

Get the curve interval affected by given control point.

```
255 {
256     return _basis.pathAffectedBy(index);
257 }
```

8.46.3.17 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 }
```

8.46.3.18 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 }
```

8.46.3.19 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 }
```

8.46.3.20 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 }
```

8.46.3.21 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 }
```

8.46.3.22 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 }
```

8.46.3.23 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 }
```

8.46.3.24 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 }
```

8.46.3.25 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 }
```

8.46.3.26 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 }
```

8.46.3.27 HxBSplineCurve HxBSplineCurve::changeAllP (const HxPointSetR2 & *p*) const [inline]

Replace the coordinates of all control points.

```
289 {
290     return HxBSplineCurve(_basis, p);
291 }
```

8.46.3.28 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 }
```

8.46.3.29 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 }
```

8.46.3.30 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 }
```

8.46.3.31 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 }
```

8.46.3.32 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 }
```

8.46.3.33 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 }
```

8.46.3.34 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

8.47 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).

8.47.1 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.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 HxBSplineInterval::HxBSplineInterval () [inline]

Default constructor.

```
99                                      : pair<double,double>()
100 {
101     _min = 0;
102     _max = 0;
103     _wrap = 0;
104 }
```

8.47.2.2 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 }
```

8.47.2.3 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 }
```

8.47.2.4 HxBSplineInterval::~HxBSplineInterval () [inline]

Destructor.

```
128 {
129 }
```

8.47.3 Member Function Documentation

8.47.3.1 double HxBSplineInterval::begin () const [inline]

first t in interval.

```
133 {
134     return first;
135 }
```

8.47.3.2 double HxBSplineInterval::end () const [inline]

last t in interval.

```
139 {
140     return second;
141 }
```

8.47.3.3 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 }
```

8.47.3.4 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 }
```

8.47.3.5 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 }
```

8.47.3.6 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 }
```

8.47.3.7 double HxBSplineInterval::middle () const [inline]

middle t of interval.

```
184 {
185     return ratio(0.5);
186 }
```

8.47.3.8 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 }
```

8.47.3.9 int HxBSplineInterval::isClosed () const [inline]

1 if interval is wrapped.

```
199 {
200     return _wrap;
201 }
```

8.47.3.10 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 }
```

8.47.3.11 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 }
```

8.47.3.12 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**

8.48 HxClassName Struct Template Reference

Class to convert a static type (i.e.

`#include <HxClassName.h>`

Public Methods

- **operator HxString** ()
  Convert to string.

8.48.1 Detailed Description

template<class Type> struct HxClassName< Type >

Class to convert a static type (i.e.

Type) to a string by calling Type::className(). Contains template specializations for types that do not have className(), e.g. builtin types.

8.48.2 Member Function Documentation

8.48.2.1 template<class Type> HxClassName< Type >::operator HxString () [inline]

Convert to string.

```
33 {
34     return Type::className();
35 }
```

The documentation for this struct was generated from the following file:

- **HxClassName.h**

8.49 HxCnum Class Reference

Basic coordinate enumerator.

`#include <HxCnum.h>`

Public Methods

- **HxCnum** ()
  Constructor.
- **HxCnum** (**HxCoord** ∗coords)
  Constructor.
- **HxCnum** (const HxCnum &rhs)
  Copy constructor.
- HxCnum & **operator=** (const HxCnum &rhs)
  Assignment operator.
- HxCnum & **operator=** (**HxCoord** ∗coords)
  Assignment operator.
- int **x** ()
  The (current) x-coordinate.
- int **y** ()
  The (current) y-coordinate.
- int **z** ()
  The (current) z-coordinate.
- void **inc** ()
  Go to the next coordinate.
- bool **operator!=** (const HxCnum &rhs)
  Comparison operator.

8.49.1 Detailed Description

Basic coordinate enumerator.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 HxCnum::HxCnum () `[inline]`

Constructor.

```
63     : _coords(0)
64 {
65 }
```

8.49.2.2 HxCnum::HxCnum (HxCoord * *coords*) [inline]

Constructor.

```
69     : _coords(coords)
70 {
71 }
```

8.49.2.3 HxCnum::HxCnum (const HxCnum & *rhs*) [inline]

Copy constructor.

```
75     : _coords(rhs._coords)
76 {
77 }
```

8.49.3 Member Function Documentation

8.49.3.1 HxCnum & HxCnum::operator= (const HxCnum & *rhs*) [inline]

Assignment operator.

```
81 {
82     _coords = rhs._coords;
83     return *this;
84 }
```

8.49.3.2 HxCnum & HxCnum::operator= (HxCoord * *coords*) [inline]

Assignment operator.

```
88 {
89     _coords = coords;
90     return *this;
91 }
```

8.49.3.3 int HxCnum::x () [inline]

The (current) x-coordinate.

```
95 {
96     return _coords->x;
97 }
```

8.49.3.4 int HxCnum::y () [inline]

The (current) y-coordinate.

```
101 {
102     return _coords->y;
103 }
```

8.49.3.5 int HxCnum::z () [inline]

The (current) z-coordinate.

```
107 {
108     return _coords->y;
109 }
```

8.49.3.6 void HxCnum::inc () [inline]

Go to the next coordinate.

```
113 {
114     _coords++;
115 }
```

8.49.3.7 bool HxCnum::operator!= (const HxCnum & *rhs*) [inline]

Comparison operator.

```
119 {
120     return _coords != rhs._coords;
121 }
```

The documentation for this class was generated from the following file:

- **HxCnum.h**

8.50 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.

8.50.1 Detailed Description

Class definition color semantics.

8.50.2 Constructor & Destructor Documentation

8.50.2.1 HxColor::HxColor () [inline]

Default constructor.

```
102     : _value(0,0,0), _space(RGB)
103 {
104 }
```

8.50.2.2 HxColor::HxColor (HxVec3Double *color*, HxColorModel *space* = RGB) [inline]

Construction from given color in color space.

```
108     : _value(color), _space(space)
109 {
110 }
```

8.50.2.3 HxColor::HxColor (const HxColor & *rhs*) [inline]

Copy constructor.

```
114     : _value(rhs._value), _space(rhs._space)
115 {
116 }
```

8.50.3 Member Function Documentation

8.50.3.1 HxColor & HxColor::operator= (const HxColor & *rhs*) [inline]

Assignment operator.

```
120 {
121     _value = rhs._value;
122     _space = rhs._space;
123     return *this;
124 }
```

8.50.3.2 const HxVec3Double & HxColor::value () const [inline]

Return the value.

```
128 {
129     return _value;
130 }
```

8.50.3.3 HxColor HxColor::convert (const HxColorModel *space*) const [inline]

General color space convertor.

See also : **Color operations on pixel values** (p. 6).

```
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 }
```

8.50.3.4 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 }
```

8.50.3.5 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 }
```

8.50.3.6 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 }
```

8.50.3.7 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 }
```

8.50.3.8 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 }
```

8.50.3.9 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 }
```

8.50.3.10 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 }
```

8.50.3.11 int HxColor::operator== (const HxColor & *c*) const [inline]

Equal.

```
149 {
150     return (_space == c._space) && (_value == c._value);
151 }
```

8.50.3.12 int HxColor::operator!= (const HxColor & *c*) const [inline]

Not equal.

```
155 {
156     return !(*this == c);
157 }
```

8.50.3.13 STD_OSTREAM & HxColor::put (STD_OSTREAM & *os*) const [inline]

Print color on stream.

For global operator<<

```
89 {
90     return os << toString();
91 }
```

8.50.3.14 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

8.51 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. 1164).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 1145).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 1262).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.51.1 Detailed Description

Class definition complex.

8.51.2 Constructor & Destructor Documentation

8.51.2.1 HxComplex::HxComplex () [inline]

Default constructor.

```
330 {
331 }
```

8.51.2.2 HxComplex::HxComplex (double *re*, double *im*) [inline]

Conversion from native type.

```
335 {
336     _values[0] = re;
337     _values[1] = im;
338 }
```

8.51.2.3 HxComplex::HxComplex (const HxComplex & *v*) [inline]

Copy constructor.

```
342 {
343     _values[0] = v._values[0];
344     _values[1] = v._values[1];
345 }
```

8.51.3 Member Function Documentation

8.51.3.1 int HxComplex::dim () const [inline]

Dimensionality.

```
363 {
364     return 2;
365 }
```

8.51.3.2 double HxComplex::x () const [inline]

Real Value.

```
369 {
370     return _values[0];
371 }
```

8.51.3.3 double HxComplex::y () const [inline]

Imaginary Value.

```
375 {
376     return _values[1];
377 }
```

8.51.3.4 double HxComplex::getValue (int *dim*) const [inline]

Element in given dimension.

```
381 {
382     return _values[dim - 1];
383 }
```

8.51.3.5 HxComplex::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 1164).

```
27 {
28     return (int) _values[0];
29 }
```

8.51.3.6 HxComplex::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 1145).

```
32 {
33     return _values[0];
34 }
```

8.51.3.7 HxComplex::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 1281).

```
37 {
38     return HxVec2Int(int(_values[0]), int(_values[1]));
39 }
```

8.51.3.8 HxComplex::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 1262).

```
42 {
43     return HxVec2Int(_values[0], _values[1]);
44 }
```

8.51.3.9 HxComplex::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 1321).

```
47 {
48     return HxVec3Int(int(_values[0]), int(_values[1]), 0);
49 }
```

8.51.3.10 HxComplex::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 1301).

```
52 {
53     return HxVec3Double(_values[0], _values[1], 0);
54 }
```

8.51.3.11 int HxComplex::operator== (const HxComplex & *v*) const [inline]

Equal.

```
393 {
394     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]);
395 }
```

8.51.3.12 int HxComplex::operator!= (const HxComplex & *v*) const [inline]

Not equal.

```
399 {
400     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]);
401 }
```

8.51.3.13 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 }
```

8.51.3.14 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 }
```

8.51.3.15 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 }
```

8.51.3.16 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 }
```

8.51.3.17 HxComplex HxComplex::operator- () const [inline]

Negation.

```
433 {
434     return HxComplex(-_values[0], -_values[1]);
435 }
```

8.51.3.18 HxComplex HxComplex::complement () const [inline]

Complement.

```
439 {
440     return HxComplex(-_values[0], -_values[1]);
441 }
```

8.51.3.19 HxComplex HxComplex::conjugate () const [inline]

Conjugate.

```
445 {
446     return HxComplex(_values[0], -_values[1]);
447 }
```

8.51.3.20 HxScalarDouble HxComplex::abs () const [inline]

Absolute value.

```
451 {
452     return HxScalarDouble(::sqrt(_values[0]*_values[0]+_values[1]*_values[1]));
453 }
```

8.51.3.21 HxScalarDouble HxComplex::arg () const [inline]

Argument.

```
457 {
458     return HxScalarDouble(::atan2(_values[1], _values[0]));
459 }
```

8.51.3.22 HxComplex HxComplex::ceil () const [inline]

Ceiling.

```
463 {
464     return HxComplex(::ceil(_values[0]), ::ceil(_values[1]));
465 }
```

8.51.3.23 HxComplex HxComplex::floor () const [inline]

Floor.

```
469 {
470     return HxComplex(::floor(_values[0]), ::floor(_values[1]));
471 }
```

8.51.3.24 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 }
```

8.51.3.25 HxComplex HxComplex::sum () const [inline]

Sum.

```
798 {
799     return *this;
800 }
```

8.51.3.26 HxComplex HxComplex::product () const [inline]

Product.

```
804 {
805     return *this;
806 }
```

8.51.3.27 HxScalarDouble HxComplex::min () const [inline]

Minimum.

```
810 {
811     return (_values[0] < _values[1]) ? _values[0] : _values[1];
812 }
```

8.51.3.28 HxScalarDouble HxComplex::max () const [inline]

Maximum.

```
816 {
817     return (_values[0] > _values[1]) ? _values[0] : _values[1];
818 }
```

8.51.3.29 HxScalarDouble HxComplex::norm1 () const

L1 norm.

```
58 {
59     return fabs(_values[0]) + fabs(_values[1]);
60 }
```

8.51.3.30 HxScalarDouble HxComplex::norm2 () const

L2 norm.

```
64 {
65     return ::sqrt(_values[0]*_values[0] + _values[1]*_values[1]);
66 }
```

8.51.3.31 HxScalarDouble HxComplex::normInf () const

L infinity norm.

```
70 {
71     return (fabs(_values[0]) > fabs(_values[1])) ? fabs(_values[0]) :
72                                                    fabs(_values[1]);
73 }
```

8.51.3.32 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 }
```

8.51.3.33 HxComplex HxComplex::sin () const [inline]

Sine.

```
494 {
495     return HxComplex(::sin(_values[0])*::cosh(_values[1]),
496         ::cos(_values[0])*::sinh(_values[1]));
497 }
```

8.51.3.34 HxComplex HxComplex::cos () const [inline]

Cosine.

```
501 {
502     return HxComplex(::cos(_values[0])*::cosh(_values[1]),
503         -::sin(_values[0])*::sinh(_values[1]));
504 }
```

8.51.3.35 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 }
```

8.51.3.36 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 }
```

8.51.3.37 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 }
```

8.51.3.38 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 }
```

8.51.3.39 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 }
```

8.51.3.40 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 }
```

8.51.3.41 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 }
```

8.51.3.42 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 }
```

8.51.3.43 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 }
```

8.51.3.44 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 }
```

8.51.3.45 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 }
```

8.51.3.46 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 }
```

8.51.3.47 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 }
```

8.51.3.48 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 }
```

8.51.3.49 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 }
```

8.51.3.50 HxComplex HxComplex::min (const HxComplex & *v*) const [inline]

Minimum.

```
679 {
680     return (operator<(v)) ? (*this) : v;
681 }
```

8.51.3.51 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 }
```

8.51.3.52 HxComplex HxComplex::max (const HxComplex & *v*) const [inline]

Maximum.

```
694 {
695     return (operator>(v)) ? (*this) : v;
696 }
```

8.51.3.53 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 }
```

8.51.3.54 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 }
```

8.51.3.55 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 }
```

8.51.3.56 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 }
```

8.51.3.57 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 }
```

8.51.3.58 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 }
```

8.51.3.59 HxComplex HxComplex::mod (const HxComplex & *v*) const [inline]

Modulo.

```
756 {
757     return (*this);
758 }
```

8.51.3.60 HxComplex HxComplex::and (const HxComplex & *v*) const [inline]

And.

```
762 {
763     return (*this);
764 }
```

8.51.3.61 HxComplex HxComplex::or (const HxComplex & *v*) const [inline]

Or.

```
768 {
769     return (*this);
770 }
```

8.51.3.62 HxComplex HxComplex::xor (const HxComplex & *v*) const [inline]

Xor.

```
774 {
775     return (*this);
776 }
```

8.51.3.63 HxComplex HxComplex::leftShift (const HxComplex & *v*) const [inline]

Left shift.

```
780 {
781     return (*this);
782 }
```

8.51.3.64 HxComplex HxComplex::rightShift (const HxComplex & *v*) const [inline]

Right shift.

```
786 {
787     return (*this);
788 }
```

8.51.3.65 HxScalarDouble HxComplex::dot (const HxComplex & *v*) const

Dot product.

```
90 {
91     return (_values[0] * v._values[0]) + (_values[1] * v._values[1]);
92 }
```

8.51.3.66 HxComplex HxComplex::cross (const HxComplex & *v*) const [inline]

Cross product.

```
792 {
793     return HxComplex(0, 0);
794 }
```

8.51.3.67 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 }
```

8.51.3.68 HxString HxComplex::toString () const

Value as a string.

```
101                          {
102     return makeString(_values[0]) + (_values[1]>=0 ? "+" : "")
103             + makeString(_values[1]) + "i";
104 }
```

8.51.4 Friends And Related Function Documentation

8.51.4.1 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 }
```

8.51.4.2 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 }
```

8.51.4.3 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 }
```

8.51.4.4 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 }
```

8.51.5 Member Data Documentation

8.51.5.1 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.

8.51.5.2 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

8.52 HxCoord Struct Reference

Representation for integer coordinates (x,y,z).

`#include <HxCnum.h>`

Public Attributes

- int **x**
- int **y**
- int **z**

8.52.1 Detailed Description

Representation for integer coordinates (x,y,z).

The documentation for this struct was generated from the following file:

- **HxCnum.h**

8.53 HxDataPtr2dScalarTem Class Template Reference

Template class for data pointers to 2D images with scalar values.

`#include <HxDataPtr2dScalarTem.h>`

Public Methods

- **HxDataPtr2dScalarTem** (PixelT ∗data, int width)
- **HxDataPtr2dScalarTem** (PixelT ∗data, size_t ∗size)
- **HxDataPtr2dScalarTem** (const HxDataPtr2dScalarTem &rhs)

- HxDataPtr2dScalarTem & **operator=** (const HxDataPtr2dScalarTem &rhs)
- void **incX** ()
- void **decX** ()
- void **incY** ()
- void **decY** ()
- void **incZ** ()
- void **decZ** ()
- void **incX** (int off)
- void **decX** (int off)
- void **incY** (int off)
- void **decY** (int off)
- void **incXYZ** (int xOff, int yOff, int zOff=0)
- void **decXYZ** (int xOff, int yOff, int zOff=0)
- ArithT **read** ()
- void **write** (const ArithT &val)
- ArithT **readIncX** ()
- void **writeIncX** (const ArithT &val)
- PixelT * **data** ()
- void **vprint** ()
- void **vprint** (HxDataPtr2dScalarTem reference)

8.53.1 Detailed Description

template<class PixelT, class ArithT> class HxDataPtr2dScalarTem< PixelT, ArithT >

Template class for data pointers to 2D images with scalar values.

The documentation for this class was generated from the following files:

- **HxDataPtr2dScalarTem.h**
- HxDataPtr2dScalarTem.c

8.54 HxDataPtr2dTem Class Template Reference

Template class for data pointers to 2D images with vector values.

`#include <HxDataPtr2dTem.h>`

Public Methods

- **HxDataPtr2dTem** (PixelT *data, int width)
- **HxDataPtr2dTem** (PixelT *data, size_t *size)
- **HxDataPtr2dTem** (const HxDataPtr2dTem &rhs)
- HxDataPtr2dTem & **operator=** (const HxDataPtr2dTem &rhs)
- void **incX** ()
- void **decX** ()
- void **incY** ()
- void **decY** ()
- void **incZ** ()

- void **decZ** ()
- void **incX** (int off)
- void **decX** (int off)
- void **incY** (int off)
- void **decY** (int off)
- void **incXYZ** (int xOff, int yOff, int zOff=0)
- void **decXYZ** (int xOff, int yOff, int zOff=0)
- ArithT **read** ()
- void **write** (const ArithT &val)
- ArithT **readIncX** ()
- void **writeIncX** (const ArithT &val)
- PixelT * **data** ()
- void **vprint** ()
- void **vprint** (HxDataPtr2dTem reference)

8.54.1 Detailed Description

template<class PixelT, class ArithT> class HxDataPtr2dTem< PixelT, ArithT >

Template class for data pointers to 2D images with vector values.

The documentation for this class was generated from the following files:

- **HxDataPtr2dTem.h**
- HxDataPtr2dTem.c

8.55 HxDataPtr3dScalarTem Class Template Reference

Template class for data pointers to 3D images with scalar values.

`#include <HxDataPtr3dScalarTem.h>`

Public Methods

- **HxDataPtr3dScalarTem** (PixelT *data, int width, int height)
- **HxDataPtr3dScalarTem** (PixelT *data, size_t *size)
- **HxDataPtr3dScalarTem** (const HxDataPtr3dScalarTem &rhs)
- HxDataPtr3dScalarTem & **operator=** (const HxDataPtr3dScalarTem &rhs)
- void **incX** ()
- void **decX** ()
- void **incY** ()
- void **decY** ()
- void **incZ** ()
- void **decZ** ()
- void **incX** (int off)
- void **decX** (int off)
- void **incY** (int off)
- void **decY** (int off)
- void **incZ** (int off)

- void **decZ** (int off)
- void **incXYZ** (int xOff, int yOff, int zOff)
- void **decXYZ** (int xOff, int yOff, int zOff)
- ArithT **read** ()
- void **write** (const ArithT &val)
- ArithT **readIncX** ()
- void **writeIncX** (const ArithT &val)
- void **vprint** ()
- PixelT * **data** ()

8.55.1 Detailed Description

template<class PixelT, class ArithT> class HxDataPtr3dScalarTem< PixelT, ArithT >

Template class for data pointers to 3D images with scalar values.

The documentation for this class was generated from the following files:

- **HxDataPtr3dScalarTem.h**
- HxDataPtr3dScalarTem.c

8.56 HxDiyTranspose Class Template Reference

Functor for transpose.

Public Methods

- **HxDiyTranspose** (**HxTagList** &)
  Constructor : empty.
- void **doIt** (DstDataPtrT dstPtr, SrcDataPtrT srcPtr, **HxSizes** dstSize, **HxSizes** srcSize)
  Actual operation.

Static Public Methods

- **HxString className** ()
  The name : "transpose".

8.56.1 Detailed Description

template<class DstDataPtrT, class SrcDataPtrT> class HxDiyTranspose< DstDataPtrT, SrcDataPtrT >

Functor for transpose.

8.56.2 Constructor & Destructor Documentation

8.56.2.1 template<class DstDataPtrT, class SrcDataPtrT> HxDiyTranspose< DstDataPtrT, SrcDataPtrT >::HxDiyTranspose (HxTagList &) [inline]

Constructor : empty.

```
28                          {}
```

8.56.3 Member Function Documentation

8.56.3.1 template<class DstDataPtrType, class SrcDataPtrType> void HxDiyTranspose< DstDataPtrType, SrcDataPtrType >::doIt (DstDataPtrType *dstPtr*, SrcDataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*)

Actual operation.

```
56 {
57     // write size check!
58     int y = dstSize.x();
59     while (--y >= 0) {
60         HxTranspose_Line(dstPtr, srcPtr, dstSize.y());
61         srcPtr.incY();
62         dstPtr.incX();
63     }
64 }
```

8.56.3.2 template<class DstDataPtrT, class SrcDataPtrT> HxString HxDiyTranspose< DstDataPtrT, SrcDataPtrT >::className () [inline, static]

The name : ”transpose”.

```
36                          { return HxString("transpose"); }
```

The documentation for this class was generated from the following file:

- **HxTranspose.c**

8.57 HxExportExtraIdentMaskCentralMoments Class Template Reference

Functor for computation of the central moments up to order N of all values in an image identified by a mask.

`#include <HxExportExtraIdentMaskCentralMoments.h>`

Public Types

- typedef **HxTagTransVar TransVarianceCategory**

  Functor is translation variant.

- typedef **HxTagNPhase PhaseCategory**

  N phases.

Public Methods

- **HxExportExtraIdentMaskCentralMoments** (**HxTagList** &tags)

  Constructor.

- ∼**HxExportExtraIdentMaskCentralMoments** ()

  Destructor.

- int **nrPhases** () const

  The number of phases.

- void **init** (int phase)

  Start of given phase.

- void **doIt** (const ImValT &imV, const ExtraValT &extraV, int x, int y, int z)

  Processing one pixel.

- void **done** (int phase)

  End of given phase.

Static Public Methods

- **HxString className** ()

  The name : "identMaskCentralMoments".

8.57.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >

Functor for computation of the central moments up to order N of all values in an image identified by a mask.

```
*  Upq = Sum[ (x - xBar)^p * (y - yBar)^q * f(x,y) ]
*         x,y
*
*  with xBar = M10 / M00 (moments!)
*  and  yBar = M01 / M00 (moments!)
*  M00 = Sum [f(x,y)], M10 = Sum [x * f(x,y)], M01 = Sum [y * f(x,y)]
*
*  The order in the resulting list is determined by:
*
*  for (int q=0 ; q<=N ; q++)
*      for (int p=0 ; p<=N ; p++)
```

```
*          if (p+q <= N)
*              [Upq];
*
*  For N=1 the order is: U00, U10, U01
*  For N=2 the order is: U00, U10, U20, U01, U11, U02
*  For N=3 the order is: U00, U10, U20, U30, U01, U11, U21, U02, U12, U03.
*
```

8.57.2 Member Typedef Documentation

8.57.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransVar HxExportExtraIdentMaskCentralMoments::TransVarianceCategory

Functor is translation variant.

8.57.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagNPhase HxExportExtraIdentMaskCentralMoments::PhaseCategory

N phases.

8.57.3 Constructor & Destructor Documentation

8.57.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::HxExportExtraIdentMaskCentralMoments (HxTagList & *tags*)

Constructor.

```
92                                                      : _tags(tags)
93 {
94     _maskVal = HxGetTag<int>(tags, "maskVal");
95     _order = HxGetTag<int>(tags, "order");
96     _valList = HxGetTag<HxValueList*>(tags, "valList");
97     _number = ((_order+1)*(_order+2))/2;
98     _sums = new ResultT[_number];
99     ResultT *ptr = _sums;
100     for (int i=0 ; i<_number; i++)
101         *ptr++ = HxScalarInt(0);
102 }
```

8.57.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::~HxExportExtraIdentMaskCentralMoments ()

Destructor.

```
107 {
108     HxValueListBackInserter res = std::back_inserter(*_valList);
109     for (int i=0 ; i<_number ; i++)
110         *res++ = HxValue(_sums[i]);
111     delete _sums;
112 }
```

8.57.4 Member Function Documentation

8.57.4.1 template<class ResultT, class ImValT, class ExtraValT> int HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::nrPhases () const [inline]

The number of phases.

```
117 {
118     return 2;
119 }
```

8.57.4.2 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::init (int *phase*) [inline]

Start of given phase.

```
124 {
125     _curPhase = phase;
126     if (phase == 1) {
127         _m00 = HxScalarInt(0);
128         _m10 = HxScalarInt(0);
129         _m01 = HxScalarInt(0);
130     } else {
131         _xBar = _m10 / _m00;
132         _yBar = _m01 / _m00;
133     }
134 }
```

8.57.4.3 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*, int *x*, int *y*, int *z*) [inline]

Processing one pixel.

```
140 {
141     if (extraV != _maskVal)
142         return;
143     if (_curPhase == 1) {
144         _m00 += imV;
145         _m10 += ResultT(HxScalarDouble(x)) * ResultT(imV);
146         _m01 += ResultT(HxScalarDouble(y)) * ResultT(imV);
147     } else {
148         ResultT xx = ResultT(HxScalarDouble(x)) - _xBar;
149         ResultT yy = ResultT(HxScalarDouble(y)) - _yBar;
150         int i=0;
151         for (int q=0 ; q<=_order ; q++) {
152             for (int p=0 ; p<=_order ; p++) {
153                 if (p+q <= _order) {
154                     _sums[i] += xx.pow(HxScalarInt(p)) * yy.pow(HxScalarInt(q)) * ResultT(imV);
155                     i++;
156                 }
157             }
158         }
159     }
160 }
```

8.57.4.4 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::done (int *phase*) [inline]

End of given phase.

```
165 {
166 }
```

8.57.4.5 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraIdentMaskCentralMoments< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : "identMaskCentralMoments".

```
171 {
172     return HxString("identMaskCentralMoments");
173 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraIdentMaskCentralMoments.h**

8.58 HxExportExtraIdentMaskMean Class Template Reference

Functor for computation of the mean of all values in an image identified by a mask.

`#include <HxExportExtraIdentMaskMean.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxExportExtraIdentMaskMean** (**HxTagList** &tags)
  Constructor.
- ~**HxExportExtraIdentMaskMean** ()
  Destructor.
- void **doIt** (const ImValT &imV, const ExtraValT &extraV)
  Processing one pixel.

Static Public Methods

- **HxString className** ()

  The name : "identMaskMean".

8.58.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraIdentMaskMean< ResultT, ImValT, ExtraValT >

Functor for computation of the mean of all values in an image identified by a mask.

8.58.2 Member Typedef Documentation

8.58.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransInVar HxExportExtraIdentMaskMean::TransVarianceCategory

Functor is translation invariant.

8.58.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTag1Phase HxExportExtraIdentMaskMean::PhaseCategory

1 phase.

8.58.3 Constructor & Destructor Documentation

8.58.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskMean< ResultT, ImValT, ExtraValT >::HxExportExtraIdentMaskMean (HxTagList & *tags*)

Constructor.

```
54                                              : _tags(tags)
55 {
56     _maskVal = HxGetTag<int>(tags, "maskVal");
57     _result = HxScalarInt(0);
58     _num = 0;
59 }
```

8.58.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskMean< ResultT, ImValT, ExtraValT >::~HxExportExtraIdentMaskMean ()

Destructor.

```
63 {
64     _result = _result / (ResultT) _num;
65     HxAddTag(_tags, "result", HxValue(_result));
66 }
```

8.58.4 Member Function Documentation

8.58.4.1 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskMean< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*) [inline]

Processing one pixel.

```
72 {
73     if (extraV == _maskVal) {
74         _result += imV;
75         _num += 1;
76     }
77 }
```

8.58.4.2 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraIdentMaskMean< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : "identMaskMean".

```
82 {
83     return HxString("identMaskMean");
84 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraIdentMaskMean.h**

8.59 HxExportExtraIdentMaskMedian Class Template Reference

Functor for computation of the median of all values in an image identified by a mask.

`#include <HxExportExtraIdentMaskMedian.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxExportExtraIdentMaskMedian** (**HxTagList** &tags)
  Constructor.
- ~**HxExportExtraIdentMaskMedian** ()

Destructor.

- void **doIt** (const ImValT &imV, const ExtraValT &extraV)

  Processing one pixel.

Static Public Methods

- **HxString className** ()

  The name : "identMaskMedian".

8.59.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraIdentMaskMedian< ResultT, ImValT, ExtraValT >

Functor for computation of the median of all values in an image identified by a mask.

8.59.2 Member Typedef Documentation

8.59.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransInVar HxExportExtraIdentMaskMedian::TransVarianceCategory

Functor is translation invariant.

8.59.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTag1Phase HxExportExtraIdentMaskMedian::PhaseCategory

1 phase.

8.59.3 Constructor & Destructor Documentation

8.59.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskMedian< ResultT, ImValT, ExtraValT >::HxExportExtraIdentMaskMedian (HxTagList & *tags*)

Constructor.

```
57                                                  : _tags(tags)
58 {
59     _maskVal = HxGetTag<int>(tags, "maskVal");
60     _result = HxScalarInt(0);
61     _minVal = ImValT::LARGE_VAL;
62     _maxVal = ImValT::SMALL_VAL;
63 }
```

8.59.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskMedian< ResultT, ImValT, ExtraValT >::~HxExportExtraIdentMaskMedian ()

Destructor.

```
67 {
68     _result = ResultT(_minVal) + ResultT(_maxVal);
69     _result /= HxScalarInt(2);
70     HxAddTag(_tags, "result", HxValue(_result));
71 }
```

8.59.4 Member Function Documentation

8.59.4.1 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskMedian< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*) [inline]

Processing one pixel.

```
77 {
78     if (extraV == _maskVal) {
79         if(imV < _minVal)
80             _minVal=imV;
81         if(imV > _maxVal)
82             _maxVal=imV;
83     }
84 }
```

8.59.4.2 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraIdentMaskMedian< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : "identMaskMedian".

```
89 {
90     return HxString("identMaskMedian");
91 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraIdentMaskMedian.h**

8.60 HxExportExtraIdentMaskMoments Class Template Reference

Functor for computation of the moments up to order N of all values in an image identified by a mask.

`#include <HxExportExtraIdentMaskMoments.h>`

Public Types

- typedef **HxTagTransVar TransVarianceCategory**

  Functor is translation variant.

- typedef **HxTag1Phase PhaseCategory**

  1 phase.

Public Methods

- **HxExportExtraIdentMaskMoments** (**HxTagList** &tags)

  Constructor.

- ∼**HxExportExtraIdentMaskMoments** ()

  Destructor.

- void **doIt** (const ImValT &imV, const ExtraValT &extraV, int x, int y, int z)

  Processing one pixel.

Static Public Methods

- **HxString className** ()

  The name : "identMaskMoments".

8.60.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraIdentMaskMoments< ResultT, ImValT, ExtraValT >

Functor for computation of the moments up to order N of all values in an image identified by a mask.

```
*  Mpq = Sum[ x^p * y^q * f(x,y) ]
*        x,y
*
*  The order in the resulting list is determined by:
*  for (int q=0 ; q<=N ; q++)
*      for (int p=0 ; p<=N ; p++)
*          if (p+q <= N)
*              [Mpq];
*
*  For N=1 the order is: m00, m10, m01
*  For N=2 the order is: m00, m10, m20, m01, m11, m02
*  For N=3 the order is: m00, m10, m20, m30, m01, m11, m21, m02, m12, m03.
*
```

8.60.2 Member Typedef Documentation

8.60.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransVar HxExportExtraIdentMaskMoments::TransVarianceCategory

Functor is translation variant.

8.60.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTag1Phase HxExportExtraIdentMaskMoments::PhaseCategory

1 phase.

8.60.3 Constructor & Destructor Documentation

8.60.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskMoments< ResultT, ImValT, ExtraValT >::HxExportExtraIdentMaskMoments (HxTagList & *tags*)

Constructor.

```
72                                                  : _tags(tags)
73 {
74     _maskVal = HxGetTag<int>(tags, "maskVal");
75     _order = HxGetTag<int>(tags, "order");
76     _valList = HxGetTag<HxValueList*>(tags, "valList");
77     _number = ((_order+1)*(_order+2))/2;
78     _sums = new ResultT[_number];
79     ResultT *ptr = _sums;
80     for (int i=0 ; i<_number; i++)
81         *ptr++ = HxScalarInt(0);
82 }
```

8.60.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskMoments< ResultT, ImValT, ExtraValT >::~HxExportExtraIdentMaskMoments ()

Destructor.

```
87 {
88     HxValueListBackInserter res = std::back_inserter(*_valList);
89     for (int i=0 ; i<_number ; i++)
90         *res++ = HxValue(_sums[i]);
91     delete _sums;
92 }
```

8.60.4 Member Function Documentation

8.60.4.1 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskMoments< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*, int *x*, int *y*, int *z*) [inline]

Processing one pixel.

```
98 {
99     if (extraV == _maskVal) {
100         int i=0;
101         for (int q=0 ; q<=_order ; q++) {
102             for (int p=0 ; p<=_order ; p++) {
103                 if (p+q <= _order) {
104                     HxScalarDouble f = ::pow(x, p) * ::pow(y, q);
105                     _sums[i] += ResultT(f) * ResultT(imV);
106                     i++;
107                 }
108             }
109         }
110     }
111 }
```

8.60.4.2 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraIdentMaskMoments< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : ”identMaskMoments”.

```
116 {
117     return HxString("identMaskMoments");
118 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraIdentMaskMoments.h**

8.61 HxExportExtraIdentMaskStdev Class Template Reference

Functor for computation of the standard deviation of all values in an image identified by a mask.

`#include <HxExportExtraIdentMaskStdev.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

- typedef **HxTag1Phase PhaseCategory**

  1 phase.

Public Methods

- **HxExportExtraIdentMaskStdev** (**HxTagList** &tags)

  Constructor.

- ∼**HxExportExtraIdentMaskStdev** ()

  Destructor.

- void **doIt** (const ImValT &imV, const ExtraValT &extraV)

  Processing one pixel.

Static Public Methods

- **HxString className** ()

  The name : "identMaskStdev".

8.61.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraIdentMaskStdev< ResultT, ImValT, ExtraValT >

Functor for computation of the standard deviation of all values in an image identified by a mask.

8.61.2 Member Typedef Documentation

8.61.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransInVar HxExportExtraIdentMaskStdev::TransVarianceCategory

Functor is translation invariant.

8.61.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTag1Phase HxExportExtraIdentMaskStdev::PhaseCategory

1 phase.

8.61.3 Constructor & Destructor Documentation

8.61.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskStdev< ResultT, ImValT, ExtraValT >::HxExportExtraIdentMaskStdev (HxTagList & *tags*)

Constructor.

```
57                                              : _tags(tags)
58 {
59     _maskVal = HxGetTag<int>(tags, "maskVal");
60     _exportVariance = HxGetTag(tags, "exportVariance",false);
61     _sum = HxScalarInt(0);
62     _sumSqr = HxScalarInt(0);
63     _num = 0;
64 }
```

8.61.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskStdev< ResultT, ImValT, ExtraValT >::~HxExportExtraIdentMaskStdev ()

Destructor.

```
68 {
69     ResultT num = _num;
70     ResultT num1 = _num - 1;
71     ResultT result = (_num == 1) ? ResultT(HxScalarInt(0))
72                                  : (_sumSqr - ((_sum * _sum) / num)) / (num1);
73     if(!_exportVariance)
74         result = result.sqrt();
75     HxAddTag(_tags, "result", HxValue(result));
76 }
```

8.61.4 Member Function Documentation

8.61.4.1 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskStdev< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*) [inline]

Processing one pixel.

```
82 {
83     if (extraV == _maskVal) {
84         _sum += imV;
85         _sumSqr += imV * imV;
86         _num += 1;
87     }
88 }
```

8.61.4.2 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraIdentMaskStdev< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : "identMaskStdev".

```
93 {
94     return HxString("identMaskStdev");
95 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraIdentMaskStdev.h**

8.62 HxExportExtraIdentMaskSum Class Template Reference

Functor for computation of the sum of all values in an image identified by a mask.

`#include <HxExportExtraIdentMaskSum.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxExportExtraIdentMaskSum** (**HxTagList** &tags)
  Constructor.
- ~**HxExportExtraIdentMaskSum** ()
  Destructor.
- void **doIt** (const ImValT &imV, const ExtraValT &extraV)
  Processing one pixel.

Static Public Methods

- **HxString className** ()
  The name : "identMaskSum".

8.62.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraIdentMaskSum< ResultT, ImValT, ExtraValT >

Functor for computation of the sum of all values in an image identified by a mask.

8.62.2 Member Typedef Documentation

8.62.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransInVar HxExportExtraIdentMaskSum::TransVarianceCategory

Functor is translation invariant.

8.62.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTag1Phase HxExportExtraIdentMaskSum::PhaseCategory

1 phase.

8.62.3 Constructor & Destructor Documentation

8.62.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskSum< ResultT, ImValT, ExtraValT >::HxExportExtraIdentMaskSum (HxTagList & *tags*)

Constructor.

```
53                                              : _tags(tags)
54 {
55     _maskVal = HxGetTag<int>(tags, "maskVal");
56     _result = HxScalarInt(0);
57 }
```

8.62.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraIdentMaskSum< ResultT, ImValT, ExtraValT >::~HxExportExtraIdentMaskSum ()

Destructor.

```
61 {
62     HxAddTag(_tags, "result", HxValue(_result));
63 }
```

8.62.4 Member Function Documentation

8.62.4.1 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraIdentMaskSum< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*) [inline]

Processing one pixel.

```
69 {
70     if (extraV == _maskVal) {
71         _result += imV;
72     }
73 }
```

8.62.4.2 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraIdentMaskSum< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : ”identMaskSum”.

```
78 {
79     return HxString("identMaskSum");
80 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraIdentMaskSum.h**

8.63 HxExportExtraWeightMaskSum Class Template Reference

Functor for computation of the sum of all values in an image weighed by a mask.

`#include <HxExportExtraWeightMaskSum.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxExportExtraWeightMaskSum** (**HxTagList** &tags)
  Constructor.
- ∼**HxExportExtraWeightMaskSum** ()
  Destructor.
- void **doIt** (const ImValT &imV, const ExtraValT &extraV)
  Processing one pixel.

Static Public Methods

- **HxString className** ()
  The name : "weightMaskSum".

8.63.1 Detailed Description

template<class ResultT, class ImValT, class ExtraValT> class HxExportExtraWeightMaskSum< ResultT, ImValT, ExtraValT >

Functor for computation of the sum of all values in an image weighed by a mask.

8.63.2 Member Typedef Documentation

8.63.2.1 template<class ResultT, class ImValT, class ExtraValT> typedef HxTagTransInVar HxExportExtraWeightMaskSum::TransVarianceCategory

Functor is translation invariant.

8.63.2.2 template<class ResultT, class ImValT, class ExtraValT> typedef HxTag1Phase HxExportExtraWeightMaskSum::PhaseCategory

1 phase.

8.63.3 Constructor & Destructor Documentation

8.63.3.1 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraWeightMaskSum< ResultT, ImValT, ExtraValT >::HxExportExtraWeightMaskSum (HxTagList & *tags*)

Constructor.

```
53                                              : _tags(tags)
54 {
55     _accum = HxScalarInt(0);
56     _weight = HxScalarInt(0);
57 }
```

8.63.3.2 template<class ResultT, class ImValT, class ExtraValT> HxExportExtraWeightMaskSum< ResultT, ImValT, ExtraValT >::~HxExportExtraWeightMaskSum ()

Destructor.

```
61 {
62     ResultT result = _accum / _weight;
63     HX_COUT << "r: " << result;
64     HxAddTag(_tags, "result", HxValue(result));
65 }
```

8.63.4 Member Function Documentation

8.63.4.1 template<class ResultT, class ImValT, class ExtraValT> void HxExportExtraWeightMaskSum< ResultT, ImValT, ExtraValT >::doIt (const ImValT & *imV*, const ExtraValT & *extraV*) [inline]

Processing one pixel.

```
71 {
72     HX_COUT << "im : " << imV << ", extra: " << extraV;
73     _accum += ResultT(imV) * ResultT(extraV);
74     _weight += ResultT(extraV);
75     HX_COUT << ", accum : " << _accum << ", weight : " << _weight << STD_ENDL;
76 }
```

8.63.4.2 template<class ResultT, class ImValT, class ExtraValT> HxString HxExportExtraWeightMaskSum< ResultT, ImValT, ExtraValT >::className () [inline, static]

The name : "weightMaskSum".

```
81 {
82     return HxString("weightMaskSum");
83 }
```

The documentation for this class was generated from the following file:

- **HxExportExtraWeightMaskSum.h**

8.64 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.

compute threshold value

- int **computeIsodataThreshold** (void)
- int **computeEntropyThreshold** (void)

Output/display

- HxHistogram **convert** (**HxValueType** dataType)
  convert dataType.
- void **getTheData** (double *x, double *y)
  Fill the externally allocated buffer with data from this histogram.
- void **getTheData2** (double *x1, double *x2, double *y)
  Fill the externally allocated buffer with data from this histogram.
- void **getTheData3** (double *x1, double *x2, double *x3, double *y)
  Fill the externally allocated buffer with data from this histogram.
- 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.

8.64.1 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.

CHANGED: JMG CdB

8.64.2 Constructor & Destructor Documentation

8.64.2.1 HxHistogram::HxHistogram ()

Construct an empty histogram.

```
68                          : _data(0)
69 {
70 }
```

8.64.2.2 HxHistogram::HxHistogram (const HxHistogram & *rhs*)

Copy constructor.

```
72                                                  : _data(rhs._data)
73 {
74 }
```

8.64.2.3 HxHistogram::HxHistogram (int *dimSize*)

Construct a 1D histogram with a range from 0 to dimSize - 1 and a binWidth of 1.

```
77 {
78     _data = new HxHistogramData(REAL_VALUE, 1,
79                                 0, dimSize - 1, dimSize,
80                                 0, 0, 0,
81                                 0, 0, 0);
82 }
```

8.64.2.4 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.

```
85 {
86     _data = new HxHistogramData(REAL_VALUE, 2,
87                                 0, dimSize1 - 1, dimSize1,
88                                 0, dimSize2 - 1, dimSize2,
89                                 0, 0, 0);
90 }
```

8.64.2.5 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.

```
93 {
94     _data = new HxHistogramData(REAL_VALUE, 3,
95                                 0, dimSize1 - 1, dimSize1,
96                                 0, dimSize2 - 1, dimSize2,
97                                 0, dimSize3 - 1, dimSize3);
98 }
```

8.64.2.6 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.

```
102 {
103     _data = new HxHistogramData(dataType, dimensions,
104                                 0, dimSize1 - 1, dimSize1,
105                                 0, dimSize2 - 1, dimSize2,
106                                 0, dimSize3 - 1, dimSize3);
107 }
```

8.64.2.7 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.

```
113 {
114     _data = new HxHistogramData(dataType, dimensions,
115                                 lowBin1, highBin1, nBins1,
116                                 lowBin2, highBin2, nBins2,
117                                 lowBin3, highBin3, nBins3);
118 }
```

8.64.2.8 HxHistogram::HxHistogram (HxString *filename*)

Read a Histogram from disk.

```
125 {
126     HxString name;
127     FILE *fp;
128     IOHEADER header;
129 
130     name = filename + ".hst";
131 
132     fp = fopen(name.c_str(), "rb");
133     if (!fp)
134         _data = 0;
135     else {
136         fread(&header, sizeof(header), 1, fp);
137 
138         _data = new HxHistogramData(header.dataType, header.dimensions,
139                             header.lowBin1, header.highBin1, header.nBins1,
140                             header.lowBin2, header.highBin2, header.nBins2,
141                             header.lowBin3, header.highBin3, header.nBins3);
142 
143         if (_data->_dataType == REAL_VALUE)
144             fread(_data->_bins.realValue, _data->_totSize,
145                 sizeof(*(_data->_bins.realValue)), fp);
146         else
147             fread(_data->_bins.intValue, _data->_totSize,
148                 sizeof(*(_data->_bins.intValue)), fp);
149         fclose(fp);
150     }
151 }
```

8.64.2.9 HxHistogram::~HxHistogram ()

Destructor.

```
154 {
155 }
```

8.64.3 Member Function Documentation

8.64.3.1 HxHistogram & HxHistogram::operator= (const HxHistogram & *rhs*)

Assignment operator.

```
166 {
167     _data = rhs._data;
168     return *this;
169 }
```

8.64.3.2 int HxHistogram::isNull () const

Indicates wether this is a valid histogram.

```
173 {
174     return _data ? 0 : 1;
175 }
```

8.64.3.3 HxHistogram::operator int () const

Indicates wether this is a valid histogram.

```
179 {
180     return _data ? 1 : 0;
181 }
```

8.64.3.4 int HxHistogram::ident () const

The unique identifier of the histogram.

```
185 {
186     return _data ? _data->_id : 0 ;
187 }
```

8.64.3.5 HxValueType HxHistogram::dataType () const

The data value type.

Either INT_VALUE or REAL_VALUE.

```
191 {
192     return _data ? _data->_dataType : INT_VALUE;
193 }
```

8.64.3.6 int HxHistogram::dimensionality () const

The number of dimensions of the histogram.

```
197 {
198     return _data ? _data->_nDims : 0;
199 }
```

8.64.3.7 int HxHistogram::dimensionSize (int *dim*) const

The size of the histogram in the i-th dimension.

The first dimension has i = 1.

```
203 {
204     if (!_data)
205         return 0;
206     switch (dim) {
207     case 1: return _data->_dimSize1;
208     case 2: return _data->_dimSize2;
209     case 3: return _data->_dimSize3;
210     }
211     return 0;
212 }
```

8.64.3.8 int HxHistogram::nrOfBins () const

The total size of the histogram.

```
216 {
217     if (!_data)
218         return 0;
219     return _data->_totSize;
220 }
```

8.64.3.9 double HxHistogram::lowBin (int *dim*) const

The lowest bin in the i-th dimension.

```
224 {
225     if (!_data)
226         return 0;
227     switch (dim) {
228     case 1: return _data->_lowBin1;
229     case 2: return _data->_lowBin2;
230     case 3: return _data->_lowBin3;
231     }
232     return 0;
233 }
```

8.64.3.10 double HxHistogram::highBin (int *dim*) const

The highest bin in the i-th dimension.

```
237 {
238     if (!_data)
239         return 0;
240     switch (dim) {
241     case 1: return _data->_highBin1;
242     case 2: return _data->_highBin2;
243     case 3: return _data->_highBin3;
244     }
245     return 0;
246 }
```

8.64.3.11 double HxHistogram::binWidth (int *dim*) const

The bin width in the i-th dimension.

```
250 {
251     if (!_data)
252         return 0;
253     switch (dim) {
254     case 1: return _data->_binWidth1;
255     case 2: return _data->_binWidth2;
256     case 3: return _data->_binWidth3;
257     }
258     return 0;
259 }
```

8.64.3.12 double HxHistogram::binToValue (int *bin*, int *dimension*) const

Translate bin to value.

```
263 {
264     if (!_data)
265         return 0;
266     switch (dimension) {
267     case 1: return (bin+0.5) * _data->_binWidth1 + _data->_lowBin1;
268     case 2: return (bin+0.5) * _data->_binWidth2 + _data->_lowBin2;
269     case 3: return (bin+0.5) * _data->_binWidth3 + _data->_lowBin3;
270     }
271     return 0;
272 }
```

8.64.3.13 int HxHistogram::valueToBin (double *val*, int *dimension*) const

Translate value to bin.

```
276 {
277     if (!_data)
278         return -1;
279     switch (dimension) {
280     case 1: return (int) ((val - _data->_lowRange1) * _data->_binFac1);
281     case 2: return (int) ((val - _data->_lowRange2) * _data->_binFac2);
282     case 3: return (int) ((val - _data->_lowRange3) * _data->_binFac3);
283     }
284     return -1;
285 }
```

8.64.3.14 double HxHistogram::get (int *bin1*) const [inline]

Get the number of elements in the given bin.

```
464 {
465     return _data ? _data->getBin(bin1) : 0;
466 }
```

8.64.3.15 double HxHistogram::get (int *bin1*, int *bin2*) const [inline]

Get the number of elements in the given bin.

```
470 {
471     return _data ? _data->getBin(bin2 * _data->_dimSize1 + bin1) : 0;
472 }
```

8.64.3.16 double HxHistogram::get (int *bin1*, int *bin2*, int *bin3*) const [inline]

Get the number of elements in the given bin.

```
476 {
477     return _data ? _data->getBin((bin3 * _data->_dimSize2 + bin2) *
478                                  _data->_dimSize1 + bin1) : 0;
479 }
```

8.64.3.17 void HxHistogram::insertValChecked (int *val*)

Insert value in 1D histogram.

```
289 {
290     if (_data && (_data->_nDims == 1))
291         incBinChecked(
292             (int) ((val - _data->_lowRange1) * _data->_binFac1));
293 }
```

8.64.3.18 void HxHistogram::insertValChecked (double *val*)

Insert value in 1D histogram.

```
297 {
298     if (_data && (_data->_nDims == 1))
299         incBinChecked(
300             (int) ((val - _data->_lowRange1) * _data->_binFac1));
301 }
```

8.64.3.19 void HxHistogram::insertValChecked (HxScalarInt *val*)

Insert value in 1D histogram.

```
305 {
306     if (_data && (_data->_nDims == 1))
307         incBinChecked(
308             (int) ((val.x() - _data->_lowRange1) * _data->_binFac1));
309 }
```

8.64.3.20 void HxHistogram::insertValChecked (HxScalarDouble *val*)

Insert value in 1D histogram.

```
313 {
314     if (_data && (_data->_nDims == 1))
315         incBinChecked(
316             (int) ((val.x() - _data->_lowRange1) * _data->_binFac1));
317 }
```

8.64.3.21 void HxHistogram::insertValChecked (HxVec2Int *val*)

Insert value in 2D histogram.

```
321 {
322     if (_data && (_data->_nDims == 2))
323         incBinChecked(
324             (int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
325             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2));
326 }
```

8.64.3.22 void HxHistogram::insertValChecked (HxVec2Double *val*)

Insert value in 2D histogram.

```
330 {
331     if (_data && (_data->_nDims == 2))
332         incBinChecked(
333             (int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
334             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2));
335 }
```

8.64.3.23 void HxHistogram::insertValChecked (HxVec3Int *val*)

Insert value in 3D histogram.

```
339 {
340     if (_data && (_data->_nDims == 3))
341         incBinChecked(
342             (int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
343             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2),
344             (int) ((val.z() - _data->_lowRange3) * _data->_binFac3));
345 }
```

8.64.3.24 void HxHistogram::insertValChecked (HxVec3Double *val*)

Insert value in 3D histogram.

```
349 {
350     if (_data && (_data->_nDims == 3))
351         incBinChecked(
352             (int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
353             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2),
354             (int) ((val.z() - _data->_lowRange3) * _data->_binFac3));
355 }
```

8.64.3.25 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.

```
359 {
360     if (_data) {
361         if ((_data->_nDims == 1) &&
362                 (bin >= 0) && (bin < _data->_dimSize1)) {
363             _data->getUnshared();
364             _data->incBin(bin);
365         }
366     }
367 }
```

8.64.3.26 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.

```
371 {
372     if (_data) {
373         if ((_data->_nDims == 2) &&
374                 (bin1 >= 0) && (bin1 < _data->_dimSize1) &&
375                 (bin2 >= 0) && (bin2 < _data->_dimSize2)) {
376             _data->getUnshared();
377             _data->incBin(bin2 * _data->_dimSize1 + bin1);
378         }
379     }
380 }
```

8.64.3.27 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.

```
384 {
385     if (_data) {
386         if ((_data->_nDims == 3) &&
387                 (bin1 >= 0) && (bin1 < _data->_dimSize1) &&
388                 (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
389                 (bin3 >= 0) && (bin3 < _data->_dimSize3)) {
390             _data->getUnshared();
391             _data->incBin((bin3 * _data->_dimSize2 + bin2) *
392                           _data->_dimSize1 + bin1);
393         }
394     }
395 }
```

8.64.3.28 void HxHistogram::insertVal (int *val*) [inline]

Insert value in 1D histogram.

```
558 {
559     incBin((int) ((val - _data->_lowRange1) * _data->_binFac1));
560 }
```

8.64.3.29 void HxHistogram::insertVal (double *val*) [inline]

Insert value in 1D histogram.

```
564 {
565     incBin((int) ((val - _data->_lowRange1) * _data->_binFac1));
566 }
```

8.64.3.30 void HxHistogram::insertVal (double *val*, double *sigma*)

Kernel Density Estimator.

```
400 {
401     if (!_data || (_data->_dataType != REAL_VALUE))
402         return;
403 
404     double ori = (val - _data->_lowBin1) / _data->_binWidth1;
405     double s = sigma/_data->_binWidth1;
406     double t, sat;
407 
408     int binmin = (int) (ori-3*s+0.5);
409     int binmax = (int) (ori+3*s+0.5);
410     int bin;
411 
412     if (binmin < 0)
413         binmin = 0;
414 
415     /* gaussian mass left outside kernel */
416     t = (ori-binmin+0.5)/(sqrt(2.0)*s);
417 
418     sat = erfc(t)*0.5;
419 
420     _data->_bins.realValue[binmin ? binmin-1 : 0] += sat;
421 
422     if (binmax >= _data->_dimSize1)
423         binmax = _data->_dimSize1-1;
424 
425     /* gaussian mass right outside kernel */
426     t = (binmax-ori+0.5)/(sqrt(2.0)*s);
427 
428     sat = erfc(t)*0.5;
429 
430     _data->_bins.realValue[binmax < _data->_dimSize1-1 ? binmax+1 : binmax]
431         += sat;
432 
433     double a = 1./(sqrt(2*M_PI)*s);
434     double b = -0.5/(s*s);
435 
436     for (bin=binmin; bin<=binmax; bin++) {
437         double v = bin-ori;
```

```
438         _data->_bins.realValue[bin] += a*exp(b*v*v);
439     }
440 }
```

8.64.3.31 void HxHistogram::insertVal (HxScalarInt *val*) [inline]

Insert value in 1D histogram.

```
570 {
571     incBin((int) ((val.x() - _data->_lowRange1) * _data->_binFac1));
572 }
```

8.64.3.32 void HxHistogram::insertVal (HxScalarDouble *val*) [inline]

Insert value in 1D histogram.

```
576 {
577     incBin((int) ((val.x() - _data->_lowRange1) * _data->_binFac1));
578 }
```

8.64.3.33 void HxHistogram::insertVal (HxVec2Int *val*) [inline]

Insert value in 2D histogram.

```
582 {
583     incBin((int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
584             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2));
585 }
```

8.64.3.34 void HxHistogram::insertVal (HxVec2Double *val*) [inline]

Insert value in 2D histogram.

```
589 {
590     incBin((int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
591             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2));
592 }
```

8.64.3.35 void HxHistogram::insertVal (HxVec3Int *val*) [inline]

Insert value in 3D histogram.

```
596 {
597     incBin((int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
598             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2),
599             (int) ((val.z() - _data->_lowRange3) * _data->_binFac3));
600 }
```

8.64.3.36 void HxHistogram::insertVal (HxVec3Double *val*) [inline]

Insert value in 3D histogram.

```
604 {
605     incBin((int) ((val.x() - _data->_lowRange1) * _data->_binFac1),
606             (int) ((val.y() - _data->_lowRange2) * _data->_binFac2),
607             (int) ((val.z() - _data->_lowRange3) * _data->_binFac3));
608 }
```

8.64.3.37 void HxHistogram::incBin (int *bin*) [inline]

Increment the given bin (assumes 1D histogram).

Values outside the histogram range are ignored.

```
483 {
484     if ((bin >= 0) && (bin < _data->_dimSize1))
485         _data->incBin(bin);
486 }
```

8.64.3.38 void HxHistogram::incBin (int *bin1*, int *bin2*) [inline]

Increment the given bin (assumes 2D histogram).

Values outside the histogram range are ignored.

```
490 {
491     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
492             (bin2 >= 0) && (bin2 < _data->_dimSize2))
493         _data->incBin(bin2 * _data->_dimSize1 + bin1);
494 }
```

8.64.3.39 void HxHistogram::incBin (int *bin1*, int *bin2*, int *bin3*) [inline]

Increment the given bin (assumes 3D histogram).

Values outside the histogram range are ignored.

```
498 {
499     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
500             (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
501             (bin3 >= 0) && (bin3 < _data->_dimSize3))
502         _data->incBin((bin3 * _data->_dimSize2 + bin2) *
503                       _data->_dimSize1 + bin1);
504 }
```

8.64.3.40 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.

```
508 {
509     if ((bin1 >= 0) && (bin1 < _data->_dimSize1))
510         _data->setBin(bin1, val);
511 }
```

8.64.3.41 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.

```
515 {
516     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
517             (bin2 >= 0) && (bin2 < _data->_dimSize2))
518         _data->setBin(bin2 * _data->_dimSize1 + bin1, val);
519 }
```

8.64.3.42 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.

```
523 {
524     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
525             (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
526             (bin3 >= 0) && (bin3 < _data->_dimSize3))
527         _data->setBin((bin3 * _data->_dimSize2 + bin2) *
528                       _data->_dimSize1 + bin1, val);
529 }
```

8.64.3.43 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.

```
533 {
534     if ((bin1 >= 0) && (bin1 < _data->_dimSize1))
535         _data->setBin(bin1, val);
536 }
```

8.64.3.44 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.

```
540 {
541     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
542             (bin2 >= 0) && (bin2 < _data->_dimSize2))
543         _data->setBin(bin2 * _data->_dimSize1 + bin1, val);
544 }
```

8.64.3.45 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.

```
548 {
549     if ((bin1 >= 0) && (bin1 < _data->_dimSize1) &&
550             (bin2 >= 0) && (bin2 < _data->_dimSize2) &&
551             (bin3 >= 0) && (bin3 < _data->_dimSize3))
552         _data->setBin((bin3 * _data->_dimSize2 + bin2) *
553                       _data->_dimSize1 + bin1, val);
554 }
```

8.64.3.46 HxHistogram HxHistogram::smooth (double *sigma* = 3.0)

Smooth histogram data.

```
444 {
445     if (!_data)
446         return HxHistogram();
447
448     GaussIIR g(sigma);
449     HxHistogramData *data = new HxHistogramData(REAL_VALUE, *_data);
450
451     /* for now only works for 1D histogram */
452     /* will be fixed when image processing works on sampled density fields */
453     /* instead of HxImageRep only... */
454
455 #ifndef _DEBUG
456     g.lineFilter(data->_bins.realValue, data->_totSize);
457 #endif
458
459     return HxHistogram(data);
460 }
```

8.64.3.47 std::list< HxVec2Double > HxHistogram::modes ()

Get histogram modes.

```
502 {
503     HxHistogramData *data = (_data->_dataType != REAL_VALUE) ?
504         new HxHistogramData(REAL_VALUE, *_data) : _data.pointee();
505
506
507
508     /* for now only works for 1D histogram */
509     /* will be fixed when image processing works on sampled density fields */
510     /* instead of HxImageRep only... */
511     /* then, see PAMI 22(11), pp. 1318--1323, 2000 */
512
513     data->getUnshared();
514     std::list<MicrosoftListSortDoesntWork> l;
515
516     /* maxima detection */
517     double *ptr = data->_bins.realValue;
518     double max = *ptr++;
519     int dir = 1;
```

```
520     for (int i=1; i < data->_totSize; i++) {
521         double val = *ptr++;
522         double grad = dir ? val-max : max-val;
523         if (grad <= 0) {
524             if (dir == 1) // maximum
525                 l.push_back(MicrosoftListSortDoesntWork(binToValue(i-1,1), max));
526             // else minimum
527             dir = dir ? 0 : 1;
528         }
529         max = val;
530     }
531 
532     if (data != _data.pointee())
533         delete data;
534 
535     l.sort();
536     std::list<HxVec2Double> res;
537     for (std::list<MicrosoftListSortDoesntWork>::iterator it = l.begin();
538             it != l.end(); it++)
539         res.push_back(HxVec2Double(*it));
540 
541     return res;
542 }
```

8.64.3.48 HxHistogram HxHistogram::normalize (double *weight* = 1.0)

Normalize histogram data.

```
546 {
547     if (!_data)
548         return HxHistogram();
549     HxHistogramData *data = new HxHistogramData(REAL_VALUE, *_data);
550 
551     int n = data->_totSize;
552     double norm = sum() / weight;
553     if (fabs(norm) > 0.0)
554     {
555         while (--n >= 0)
556             data->_bins.realValue[n] /= norm;
557         data->_sum = weight;
558     }
559 
560     return HxHistogram(data);
561 }
```

8.64.3.49 double HxHistogram::sum () const

The sum of the histogram.

```
565 {
566     if (!_data)
567         return 0;
568     int n = _data->_totSize;
569     double s = 0;
570     if (_data->_dataType == REAL_VALUE)
571         while (--n >= 0)
572             s += _data->_bins.realValue[n];
573     else
574         while (--n >= 0)
```

```
575             s += _data->_bins.intValue[n];
576     _data->_sum = s;
577     return s;
578 }
```

8.64.3.50 double HxHistogram::minVal () const

The minimum number of elements in the histogram.

```
582 {
583     if (!_data)
584         return 0;
585     int n = _data->_totSize;
586     double m;
587     if (_data->_dataType == REAL_VALUE) {
588         m = _data->_bins.realValue[0];
589         while (--n >= 0)
590             if (_data->_bins.realValue[n] < m)
591                 m = _data->_bins.realValue[n];
592     }
593     else {
594         m = _data->_bins.intValue[0];
595         while (--n >= 0)
596             if (_data->_bins.intValue[n] < m)
597                 m = _data->_bins.intValue[n];
598     }
599     return m;
600 }
```

8.64.3.51 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.

```
626 {
627     if (!_data)
628         return 0;
629     int n = _data->_totSize;
630     double m;
631     *index = 0;
632     if (_data->_dataType == REAL_VALUE) {
633         m = _data->_bins.realValue[0];
634         while (--n >= 0)
635             if (_data->_bins.realValue[n] < m) {
636                 m = _data->_bins.realValue[n];
637                 *index = n;
638             }
639     }
640     else {
641         m = _data->_bins.intValue[0];
642         while (--n >= 0)
643             if (_data->_bins.intValue[n] < m) {
644                 m = _data->_bins.intValue[n];
645                 *index = n;
646             }
647     }
648     return m;
649 }
```

8.64.3.52 double HxHistogram::maxVal () const

The maximum number of elements in the histogram.

```
604 {
605     if (!_data)
606         return 0;
607     int n = _data->_totSize;
608     double m;
609     if (_data->_dataType == REAL_VALUE) {
610         m = _data->_bins.realValue[0];
611         while (--n >= 0)
612             if (_data->_bins.realValue[n] > m)
613                 m = _data->_bins.realValue[n];
614     }
615     else {
616         m = _data->_bins.intValue[0];
617         while (--n >= 0)
618             if (_data->_bins.intValue[n] > m)
619                 m = _data->_bins.intValue[n];
620     }
621     return m;
622 }
```

8.64.3.53 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.

```
653 {
654     if (!_data)
655         return 0;
656     int n = _data->_totSize;
657     double m;
658     *index = 0;
659     if (_data->_dataType == REAL_VALUE) {
660         m = _data->_bins.realValue[0];
661         while (--n >= 0)
662             if (_data->_bins.realValue[n] > m) {
663                 m = _data->_bins.realValue[n];
664                 *index = n;
665             }
666     }
667     else {
668         m = _data->_bins.intValue[0];
669         while (--n >= 0)
670             if (_data->_bins.intValue[n] > m) {
671                 m = _data->_bins.intValue[n];
672                 *index = n;
673             }
674     }
675     return m;
676 }
```

8.64.3.54 double HxHistogram::intersection (const HxHistogram & *rhs*) const

Intersection of histograms.

```
700 {
701     if (!_data)
702         return 0;
703     if (!isEqualSize(rhs))
704         return -1;
705     int n = _data->_totSize;
706     double s = 0;
707     if (_data->_dataType == REAL_VALUE)
708         if (rhs._data->_dataType == REAL_VALUE)
709             s = ::intersect(_data->_bins.realValue, rhs._data->_bins.realValue, n);
710         else
711             s = ::intersect(_data->_bins.realValue, rhs._data->_bins.intValue, n);
712     else
713         if (rhs._data->_dataType == REAL_VALUE)
714             s = ::intersect(_data->_bins.intValue, rhs._data->_bins.realValue, n);
715         else
716             s = ::intersect(_data->_bins.intValue, rhs._data->_bins.intValue, n);
717
718     return s;
719 }
```

8.64.3.55 double HxHistogram::chiSquare (const HxHistogram & *rhs*) const

Chi square intersection of histograms.

```
737 {
738     if (!_data)
739         return 0;
740     if (!isEqualSize(rhs))
741         return -1;
742     int n = _data->_totSize;
743     double s = 0;
744     if (_data->_dataType == REAL_VALUE)
745         if (rhs._data->_dataType == REAL_VALUE)
746             s = ::chiSq(_data->_bins.realValue, rhs._data->_bins.realValue, n);
747         else
748             s = ::chiSq(_data->_bins.realValue, rhs._data->_bins.intValue, n);
749     else
750         if (rhs._data->_dataType == REAL_VALUE)
751             s = ::chiSq(_data->_bins.intValue, rhs._data->_bins.realValue, n);
752         else
753             s = ::chiSq(_data->_bins.intValue, rhs._data->_bins.intValue, n);
754     return s;
755 }
```

8.64.3.56 double HxHistogram::chiSquareNorm (const HxHistogram & *rhs*) const

Normalized chi square intersection of histograms.

```
774 {
775     if (!_data)
776         return 0;
777     if (!isEqualSize(rhs))
778         return -1;
779     int n = _data->_totSize;
780     double s = 0;
781     if (_data->_dataType == REAL_VALUE)
782         if (rhs._data->_dataType == REAL_VALUE)
783             s = ::chiSqNorm(_data->_bins.realValue, rhs._data->_bins.realValue, n);
```

```
784         else
785             s = ::chiSqNorm(_data->_bins.realValue, rhs._data->_bins.intValue, n);
786     else
787         if (rhs._data->_dataType == REAL_VALUE)
788             s = ::chiSqNorm(_data->_bins.intValue, rhs._data->_bins.realValue, n);
789         else
790             s = ::chiSqNorm(_data->_bins.intValue, rhs._data->_bins.intValue, n);
791     return s;
792 }
```

8.64.3.57 HxHistogram HxHistogram::convert (HxValueType *dataType*)

convert dataType.

```
159 {
160     HxHistogramData *data = new HxHistogramData(*_data);
161     return HxHistogram(data);
162 }
```

8.64.3.58 void HxHistogram::getTheData (double * *x*, double * *y*)

Fill the externally allocated buffer with data from this histogram.

The pointer to x represents the bin value, y the bin count.

```
796 {
797     if (!_data)
798         return;
799
800     int n = 0;
801     if (_data->_dataType == REAL_VALUE)
802         for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++) {
803             x[n] = binToValue(dim1,1);
804             y[n] = _data->_bins.realValue[dim1];
805             n++;
806         }
807     else
808         for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++) {
809             x[n] = binToValue(dim1,1);
810             y[n] = _data->_bins.intValue[dim1];
811             n++;
812         }
813 }
```

8.64.3.59 void HxHistogram::getTheData2 (double * *x1*, double * *x2*, double * *y*)

Fill the externally allocated buffer with data from this histogram.

The pointer to x1,x2 represents the bin value, y the bin count.

```
817 {
818     if (!_data)
819         return;
820
821     int n = 0;
822     if (_data->_dataType == REAL_VALUE)
```

```
823         for (int dim2=0 ; dim2<dimensionSize(2) ; dim2++) {
824             for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++) {
825                 x1[n] = binToValue(dim1,1);
826                 x2[n] = binToValue(dim2,2);
827                 y[n] = _data->_bins.realValue[dim2 * _data->_dimSize1
828                             + dim1];
829                 n++;
830             }
831         }
832     else
833         for (int dim2=0 ; dim2<dimensionSize(2) ; dim2++) {
834             for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++) {
835                 x1[n] = binToValue(dim1,1);
836                 x2[n] = binToValue(dim2,2);
837                 y[n] = _data->_bins.intValue[dim2 * _data->_dimSize1
838                             + dim1];
839                 n++;
840             }
841         }
842 }
```

8.64.3.60 void HxHistogram::getTheData3 (double * *x1*, double * *x2*, double * *x3*, double * *y*)

Fill the externally allocated buffer with data from this histogram.

The pointer to x1,x2,x3 represents the bin value, y the bin count.

```
846 {
847     if (!_data)
848         return;
849 
850     int n = 0;
851     if (_data->_dataType == REAL_VALUE)
852         for (int dim3=0 ; dim3<dimensionSize(3) ; dim3++) {
853             for (int dim2=0 ; dim2<dimensionSize(2) ; dim2++) {
854                 for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++) {
855                     x1[n] = binToValue(dim1,1);
856                     x2[n] = binToValue(dim2,2);
857                     x3[n] = binToValue(dim3,3);
858                     y[n] = _data->_bins.realValue[(dim3 * _data->_dimSize2
859                                 + dim2) * _data->_dimSize1 + dim1];
860                     n++;
861                 }
862             }
863         }
864     else
865         for (int dim3=0 ; dim3<dimensionSize(3) ; dim3++) {
866             for (int dim2=0 ; dim2<dimensionSize(2) ; dim2++) {
867                 for (int dim1=0 ; dim1<dimensionSize(1) ; dim1++) {
868                     x1[n] = binToValue(dim1,1);
869                     x2[n] = binToValue(dim2,2);
870                     x3[n] = binToValue(dim3,3);
871                     y[n] = _data->_bins.intValue[(dim3 * _data->_dimSize2
872                                 + dim2) * _data->_dimSize1 + dim1];
873                     n++;
874                 }
875             }
876         }
877 }
```

8.64.3.61 void HxHistogram::getDataDouble (double * *data*)

Fill the externally allocated buffer with data from this histogram.

```
881 {
882     if (!_data)
883         return;
884     int n = _data->_totSize;
885     if (_data->_dataType == REAL_VALUE)
886         while (--n >= 0)
887             data[n] = _data->_bins.realValue[n];
888     else
889         while (--n >= 0)
890             data[n] = _data->_bins.intValue[n];
891 }
```

8.64.3.62 void HxHistogram::getDataInt (int * *data*)

Fill the externally allocated buffer with data from this histogram.

```
895 {
896     if (!_data)
897         return;
898     int n = _data->_totSize;
899     if (_data->_dataType == REAL_VALUE)
900         while (--n >= 0)
901             data[n] = (int)(_data->_bins.realValue[n] + 0.5);
902     else
903         while (--n >= 0)
904             data[n] = _data->_bins.intValue[n];
905 }
```

8.64.3.63 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.

```
982 {
983     if (!_data)
984         return;
985     /*
986     el = (elevation * M_PI) / 180.;
987     al = (alpha * M_PI) / 180.;
988     cosa = cos(al);
989     sina = sin(al);
990     cose = cos(el);
991     sine = sin(el);
992     dataW = dataWidth;
993     xMin = 190; //for 3D histo
994     yMax = 190; //255;
995     */
996     double el = (elevation * M_PI) / 180.;
997     double al = (alpha * M_PI) / 180.;
998     double cosa = cos(al);
999     double sina = sin(al);
1000     double cose = cos(el);
```

```
1001     double sine = sin(el);
1002     int dataW = dataWidth;
1003     int xMin = 190; //for 3D histo
1004     int yMax = 190; //255;
1005     drawAxis(data,cosa,sina,cose,sine,dataW);
1006
1007     int n = 0;
1008
1009     if (_data->_dataType == REAL_VALUE) {
1010         for (int b=0 ; b < _data->_dimSize3 ; b++) {
1011             for (int g=0 ; g < _data->_dimSize2 ; g++) {
1012                 for (int r=0 ; r < _data->_dimSize1 ; r++) {
1013                     if (_data->_bins.realValue[n] > threshold) {
1014                         /*
1015                           Doen't work if hist range unequals 0..255
1016                         double rV = r * _data->_binWidth1 + _data->_lowBin1;
1017                         double gV = g * _data->_binWidth2 + _data->_lowBin2;
1018                         double bV = b * _data->_binWidth3 + _data->_lowBin3;
1019                         */
1020                         double rV = r * 255./(_data->_dimSize1-1);
1021                         double gV = g * 255./(_data->_dimSize2-1);
1022                         double bV = b * 255./(_data->_dimSize3-1);
1023                         setPixelV(data, transf3Dto2D(rV,gV,bV,cosa,sina,cose,sine), HxVec3Int(rV,gV,bV), dataW);
1024                     }
1025                     n++;
1026                 }
1027             }
1028         }
1029     }
1030     else {
1031         for (int b=0 ; b < _data->_dimSize3 ; b++) {
1032             for (int g=0 ; g < _data->_dimSize2 ; g++) {
1033                 for (int r=0 ; r < _data->_dimSize1 ; r++) {
1034                     if (_data->_bins.intValue[n] > threshold) {
1035                         /*
1036                           Doen't work if hist range unequals 0..255
1037                         double rV = r * _data->_binWidth1 + _data->_lowBin1;
1038                         double gV = g * _data->_binWidth2 + _data->_lowBin2;
1039                         double bV = b * _data->_binWidth3 + _data->_lowBin3;
1040                         */
1041                         double rV = r * 255./(_data->_dimSize1-1);
1042                         double gV = g * 255./(_data->_dimSize2-1);
1043                         double bV = b * 255./(_data->_dimSize3-1);
1044                         setPixelV(data, transf3Dto2D(rV,gV,bV,cosa,sina,cose,sine), HxVec3Int(rV,gV,bV), dataW);
1045                     }
1046                     n++;
1047                 }
1048             }
1049         }
1050     }
1051 }
```

8.64.3.64 STD_OSTREAM & HxHistogram::put (STD_OSTREAM & *os*, HxString *delimit* = " ") const

Print histogram data in the given stream.

```
1055 {
1056     if (!_data)
1057         return os << "HxHistogram(null)" << STD_ENDL;
1058
1059     int dim1, dim2, dim3;
```

```
1060     if (_data->_dataType == REAL_VALUE)
1061         switch (dimensionality()) {
1062         case 1:
1063             for (dim1=0 ; dim1<dimensionSize(1) ; dim1++)
1064                 os << binToValue(dim1,1) << delimit <<
1065                    float(_data->_bins.realValue[dim1]) << STD_ENDL;
1066             break;
1067         case 2:
1068             for (dim2=0 ; dim2<dimensionSize(2) ; dim2++)
1069                 for (dim1=0 ; dim1<dimensionSize(1) ; dim1++)
1070                     os << binToValue(dim1,1) << delimit <<
1071                        binToValue(dim2,2) << delimit <<
1072                        float(_data->_bins.realValue[dim2 * _data->_dimSize1
1073                            + dim1]) << STD_ENDL;
1074             break;
1075         case 3:
1076             for (dim3=0; dim3<dimensionSize(3) ; dim3++)
1077                 for (dim2=0 ; dim2<dimensionSize(2) ; dim2++)
1078                     for (dim1=0 ; dim1<dimensionSize(1) ; dim1++)
1079                         os << binToValue(dim1,1) << delimit <<
1080                            binToValue(dim2,2) << delimit <<
1081                            binToValue(dim3,3) << delimit <<
1082                            float(_data->_bins.realValue[(dim3 * _data->_dimSize2
1083                                 + dim2) * _data->_dimSize1 + dim1]) << STD_ENDL;
1084             break;
1085         }
1086     else
1087         switch (dimensionality()) {
1088         case 1:
1089             for (dim1=0 ; dim1<dimensionSize(1) ; dim1++)
1090                 os << binToValue(dim1,1) << delimit <<
1091                    _data->_bins.intValue[dim1] << STD_ENDL;
1092             break;
1093         case 2:
1094             for (dim2=0 ; dim2<dimensionSize(2) ; dim2++)
1095                 for (dim1=0 ; dim1<dimensionSize(1) ; dim1++)
1096                     os << binToValue(dim1,1) << delimit <<
1097                        binToValue(dim2,2) << delimit <<
1098                        _data->_bins.intValue[dim2 * _data->_dimSize1
1099                            + dim1] << STD_ENDL;
1100             break;
1101         case 3:
1102             for (dim3=0; dim3<dimensionSize(3) ; dim3++)
1103                 for (dim2=0 ; dim2<dimensionSize(2) ; dim2++)
1104                     for (dim1=0 ; dim1<dimensionSize(1) ; dim1++)
1105                         os << binToValue(dim1,1) << delimit <<
1106                            binToValue(dim2,2) << delimit <<
1107                            binToValue(dim3,3) << delimit <<
1108                            _data->_bins.intValue[(dim3 * _data->_dimSize2
1109                                 + dim2) * _data->_dimSize1 + dim1] << STD_ENDL;
1110             break;
1111         }
1112     return os;
1113 }
```

8.64.3.65 int HxHistogram::write (HxString *filename*)

Write histogram to disk.

```
1132 {
1133     HxString name;
1134     FILE *fp;
```

```
1135     IOHEADER header;
1136
1137     if (!_data)
1138         return 0;
1139
1140     name = filename + ".hst";
1141
1142     fp = fopen(name.c_str(), "wb");
1143     if (!fp)
1144         return 0;
1145
1146     header.dataType = _data->_dataType;
1147     header.dimensions = _data->_nDims;
1148     header.lowBin1 = _data->_lowBin1;
1149     header.highBin1 = _data->_highBin1;
1150     header.nBins1 = _data->_dimSize1;
1151     header.lowBin2 = _data->_lowBin2;
1152     header.highBin2 = _data->_highBin2;
1153     header.nBins2 = _data->_dimSize2;
1154     header.lowBin3 = _data->_lowBin3;
1155     header.highBin3 = _data->_highBin3;
1156     header.nBins3 = _data->_dimSize3;
1157     fwrite(&header, sizeof(header), 1, fp);
1158
1159     if (_data->_dataType == REAL_VALUE)
1160         fwrite(_data->_bins.realValue, _data->_totSize,
1161             sizeof(*(_data->_bins.realValue)), fp);
1162     else
1163         fwrite(_data->_bins.intValue, _data->_totSize,
1164             sizeof(*(_data->_bins.intValue)), fp);
1165
1166     fclose(fp);
1167
1168     return 1;
1169 }
```

8.64.3.66 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.

```
1173 {
1174     if (!_data)
1175         return HxHistogram();
1176     HxHistogramData *data = new HxHistogramData(*_data);
1177
1178     int n = data->_totSize;
1179
1180     if (data->_dataType == REAL_VALUE)
1181         while (--n >= 0) {
1182             if (data->_bins.realValue[n]<=valThreshold)
1183                 data->_bins.realValue[n]=0.0;
1184         }
1185     else
1186         while (--n >= 0) {
1187             if (data->_bins.intValue[n]<=valThreshold)
1188                 data->_bins.intValue[n]=0.0;
1189         }
1190
1191     return HxHistogram(data);
1192 }
```

8.64.3.67 int HxHistogram::countBins (double *valThreshold* = 0.0)

Returns number of bins with count>valThreshold.

Counts number of non-empty bins by default.

```
1196 {
1197     if (!_data)
1198         return 0;
1199
1200     int n = _data->_totSize;
1201
1202     int counter=0;
1203
1204     if (_data->_dataType == REAL_VALUE)
1205         while (--n >= 0) {
1206             if (_data->_bins.realValue[n]>valThreshold)
1207                counter++;
1208         }
1209     else
1210         while (--n >= 0) {
1211             if (_data->_bins.intValue[n]>valThreshold)
1212                counter++;
1213         }
1214
1215     return counter;
1216 }
```

8.64.3.68 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. 556)-1.

```
1222 {
1223     if (!_data)
1224         return HxHistogram();
1225
1226     // Default value binMax=-1 maps to dimensionSize()-1.
1227     if (binMax1<0)
1228         binMax1=dimensionSize(1)-1;
1229     if (binMax2<0)
1230         binMax2=dimensionSize(2)-1;
1231     if (binMax3<0)
1232         binMax3=dimensionSize(3)-1;
1233
1234     if (binMin1<0 || binMin2<0 || binMin3<0)
1235     {
1236         STD_CERR << "Error in HxHistogram::reduceRange, minimum smaller than 0." << STD_ENDL;
1237         return HxHistogram();
1238     }
1239
1240     if (binMax1<binMin1 || binMax2<binMin2 || binMax3<binMin3)
1241     {
1242         STD_CERR << "Error in HxHistogram::reduceRange, minimum higher than maximum." << STD_ENDL;
1243         return HxHistogram();
1244     }
1245
1246     // Make new histogram for given range
1247     HxHistogram h=HxHistogram(dataType(),dimensionality(),
```

```
1248         binToValue(binMin1,1),binToValue(binMax1,1),binMax1-binMin1+1,
1249         binToValue(binMin2,2),binToValue(binMax2,2),binMax2-binMin2+1,
1250         binToValue(binMin3,3),binToValue(binMax3,3),binMax3-binMin3+1);
1251
1252     if (_data->_dataType == REAL_VALUE) {
1253         for (int z=binMin3; z<=binMax3; z++)
1254             for (int y=binMin2; y<=binMax2; y++)
1255                 for (int x=binMin1; x<=binMax1; x++) {
1256                     int n = ((z-binMin3) * _data->_dimSize2 + (y-binMin2)) *
1257                          _data->_dimSize1 + x-binMin1;
1258                     int m = (z * _data->_dimSize2 + y) *
1259                          _data->_dimSize1 + x;
1260                     h._data->_bins.realValue[n] = _data->_bins.realValue[m];
1261                 }
1262     }
1263     else {
1264         for (int z=binMin3; z<=binMax3; z++)
1265             for (int y=binMin2; y<=binMax2; y++)
1266                 for (int x=binMin1; x<=binMax1; x++) {
1267                     int n = ((z-binMin3) * _data->_dimSize2 + (y-binMin2)) *
1268                          _data->_dimSize1 + x-binMin1;
1269                     int m = (z * _data->_dimSize2 + y) *
1270                          _data->_dimSize1 + x;
1271                     h._data->_bins.intValue[n] = _data->_bins.intValue[m];
1272                 }
1273     }
1274
1275     return h;
1276 }
```

8.64.3.69 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.

```
1281 {
1282     if (binValMin1==binValMax1) {
1283         binValMin1=_data->_lowBin1;
1284         binValMax1=_data->_highBin1;
1285     }
1286     if (binValMin1<_data->_lowBin1)
1287         binValMin1=_data->_lowBin1;
1288     if (binValMax1>_data->_highBin1)
1289         binValMax1=_data->_highBin1;
1290     if (binValMin2==binValMax2) {
1291         binValMin2=_data->_lowBin2;
1292         binValMax2=_data->_highBin2;
1293     }
1294     if (binValMin2<_data->_lowBin2)
1295         binValMin2=_data->_lowBin2;
1296     if (binValMax2>_data->_highBin2)
1297         binValMax2=_data->_highBin2;
1298     if (binValMin3==binValMax3) {
1299         binValMin3=_data->_lowBin3;
1300         binValMax3=_data->_highBin3;
1301     }
1302     if (binValMin3<_data->_lowBin3)
1303         binValMin3=_data->_lowBin3;
1304     if (binValMax3>_data->_highBin3)
```

```
1305         binValMax3=_data->_highBin3;
1306     return reduceRange(valueToBin(binValMin1,1),valueToBin(binValMax1,1),
1307         valueToBin(binValMin2,2),valueToBin(binValMax2,2),
1308         valueToBin(binValMin3,3),valueToBin(binValMax3,3));
1309 }
```

8.64.3.70 HxHistogram HxHistogram::to1D (int *dim* = 1)

Projects n-dimensional (1<n<=3) histogram on a 1-D histogram for given dimension.

```
1314 {
1315     if (dimensionality()<2)
1316     {
1317 //      STD_CERR << "Warning. HxHistogram::to1D, dimensionality is " << dimensionality() << STD_ENDL;
1318         return *this;
1319     }
1320
1321     if (dim<1 || dim>3)
1322     {
1323         STD_CERR << "Error. Dimension out of bounds: " << dim << STD_ENDL;
1324         return HxHistogram();
1325     }
1326
1327
1328     HxHistogram result;
1329     result=HxHistogram(dataType(), 1,
1330                        lowBin(dim),highBin(dim),dimensionSize(dim),
1331                        0,0,0,
1332                        0,0,0);
1333
1334     int otherDim1=-1;
1335     int otherDim2=-1;
1336
1337     if (dim==1)
1338     {
1339         otherDim1=3;
1340         otherDim2=2;
1341     }
1342     if (dim==2)
1343     {
1344         otherDim1=3;
1345         otherDim2=1;
1346     }
1347     if (dim==3)
1348     {
1349         otherDim1=2;
1350         otherDim2=1;
1351     }
1352
1353     for (int b=0; b<dimensionSize(dim); b++)
1354     {
1355         double total=0;
1356         for (int sb=0; sb<dimensionSize(otherDim1); sb++)
1357             for (int tb=0; tb<dimensionSize(otherDim2); tb++)
1358             {
1359                 if (dim==1)
1360                     total+=get(b,tb,sb);
1361                 if (dim==2)
1362                     total+=get(tb,b,sb);
1363                 if (dim==3)
1364                     total+=get(tb,sb,b);
1365             }
```

```
1366         result.setBin(b, total);
1367     }
1368
1369     return result;
1370 }
```

The documentation for this class was generated from the following files:

- **HxHistogram.h**
- HxHistogram.c

8.65 HxIfRbPair Class Reference

Representation for the result of a query of the rule base.

`#include <HxImgFtorRuleBase.h>`

Public Methods

- **HxIfRbPair** ()

  Constructor.

- **HxIfRbPair** (**HxImageSignature** s, bool f=true)

  Constructor.

- **operator int** ()

  Cast found to an integer.

- **operator HxImageSignature** ()

  Cast result to an **HxImageSignature** (p. 685).

Public Attributes

- bool **found**

  Was it really found?

- **HxImageSignature sig**

  The result.

8.65.1 Detailed Description

Representation for the result of a query of the rule base.

The result is basically an **HxImageSignature** (p. 685) annotaed with a flag ”found” telling whether the result was actually found in the rule base or was constructed from a default image signature.

8.65.2 Constructor & Destructor Documentation

8.65.2.1 HxIfRbPair::HxIfRbPair () [inline]

Constructor.

```
27 {}
```

8.65.2.2 HxIfRbPair::HxIfRbPair (HxImageSignature *s*, bool *f* = true) [inline]

Constructor.

```
31                                      : sig(s), found(f) {}
```

8.65.3 Member Function Documentation

8.65.3.1 HxIfRbPair::operator int () [inline]

Cast found to an integer.

```
39 { return found ? 1 : 0; }
```

8.65.3.2 HxIfRbPair::operator HxImageSignature () [inline]

Cast result to an **HxImageSignature** (p. 685).

```
41 { return sig; }
```

8.65.4 Member Data Documentation

8.65.4.1 bool HxIfRbPair::found

Was it really found?

8.65.4.2 HxImageSignature HxIfRbPair::sig

The result.

The documentation for this class was generated from the following file:

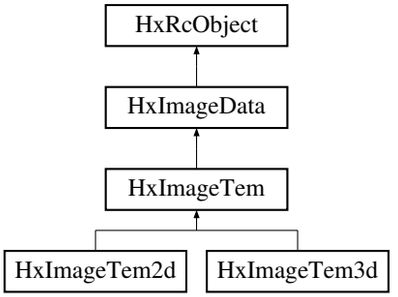
- **HxImgFtorRuleBase.h**

8.66 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

  Get identity.

- **HxString name** () const

  Get name.

- void **name** (**HxString** s)

  Set name.

- virtual int **dimensionality** () const=0

  Get dimensionality.

- virtual int **dimensionSize** (int i) const=0

  Get image size in given dimension.

- virtual **HxSizes sizes** () const=0

  Get image sizes.

- virtual int **numberOfPixels** () const=0

  Get total number of pixels.

- virtual int **pixelDimensionality** () const=0

Get dimensionality of pixels.

- virtual **HxValueType pixelType** () const=0

  Get type of pixel.

- virtual int **pixelPrecision** () const=0

  Get pixel precision.

- virtual **HxImageSignature signature** () const=0

  Get image signature.

Import/export operations

- virtual void **import** (**HxByte** ∗data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp=”importPix”)

  Import pixels from HxByte data.

- virtual void **import** (short ∗data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp=”importPix”)

  Import pixels from short data.

- virtual void **import** (int ∗data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp=”importPix”)

  Import pixels from int data.

- virtual void **import** (float ∗data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp=”importPix”)

  Import pixels from float data.

- virtual void **import** (double ∗data, **HxTagList** &tags=HxMakeTagList(), **HxString** importOp=”importPix”)

  Import pixels from double data.

- virtual void **exportOp** (**HxByte** ∗data, **HxTagList** &tags, **HxString** exportOp=”exportPix”)

  Export pixels as HxByte data.

- virtual void **exportOp** (short ∗data, **HxTagList** &tags, **HxString** exportOp=”exportPix”)

  Export pixels as short data.

- virtual void **exportOp** (int ∗data, **HxTagList** &tags, **HxString** exportOp=”exportPix”)

  Export pixels as int data.

- virtual void **exportOp** (float ∗data, **HxTagList** &tags, **HxString** exportOp=”exportPix”)

  Export pixels as float data.

- virtual void **exportOp** (double ∗data, **HxTagList** &tags, **HxString** exportOp=”exportPix”)

  Export pixels as double data.

- virtual void **inout** (void ∗data, **HxString** dataType, **HxString** inOutOp, **HxTagList** &tags)

InOut operation.

- virtual void **inout** (**HxString** inOutOp, **HxTagList** &tags)

  InOut operation (the real one).

- virtual void **exportExtra** (**HxString** exportOp, HxImageData *extraIm, **HxTagList** &tags)

  Export operation with extra image.

Set and border operations

- virtual void **set** (const **HxValue** val)
- virtual void **set** (HxImageData *arg)
- virtual void **set** (int *pixels)
- virtual void **set** (**HxByte** *pixels)
- virtual void **set** (double *pixels)=0
- void **setPartImage** (HxImageData *src)
- void **setPartImage** (HxImageData *src, **HxTagList** &tags)

  Set operation (the real one).

- void **setPartImage** (HxImageData *src, **HxPointInt** srcBegin, **HxPointInt** srcEnd, **HxPointInt** dstBegin)
- void **setBorder** (**HxTagList** &tags)

  Set border operation (the real one).

- 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** (HxImageData *arg1, HxImageData *arg2, **HxString** bpoName, **HxTagList** &tags)

  Binary pixel operation.

- virtual void **multiPixOp** (HxImageData **args, int nArgs, **HxString** mpoName, **HxTagList** &tags)

  Multi pixel operation.

- virtual void **generalizedConvolution** (HxImageData *srcImg, HxImageData *kerImg, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Generalized convolution operation.

- virtual void **generalizedConvolutionK1d** (HxImageData ∗srcImg, int dimension, HxImageData ∗kerImg, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Generalized convolution operation in one dimension.

- virtual void **genConvSeparated** (HxImageData ∗srcImg, int dimension, HxImageData ∗kerImg1, HxImageData ∗kerImg2, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Generalized convolution operation separated by dimension.

- virtual void **genConv2dSep** (HxImageData ∗srcImg, HxImageData ∗kerImg1, HxImageData ∗kerImg2, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Separable generalized convolution operation on 2D images.

- virtual void **genConv3dSep** (HxImageData ∗srcImg, HxImageData ∗kerImg1, HxImageData ∗kerImg2, HxImageData ∗kerImg3, **HxString** genMul, **HxString** genAdd, **HxString** kerName, **HxTagList** &tags)

  Separable generalized convolution operation on 3D images.

- virtual void **recGenConv** (HxImageData ∗srcImg, HxImageData ∗kerImg, **HxString** genMul, **HxString** genAdd, **HxTagList** &tags)

  Recursive generalized convolution operation.

- virtual void **recGenConv2dSep** (HxImageData ∗srcImg, HxImageData ∗kerImg1, HxImageData ∗kerImg2, **HxString** genMul, **HxString** genAdd, **HxTagList** &tags)

  Separable recursive generalized convolution operation on 2D images.

- virtual void **neighbourhoodOp** (HxImageData ∗src, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation.

- virtual void **neighbourhoodOpExtra** (HxImageData ∗src, HxImageData ∗extraIm, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation with extra image.

- virtual void **neighbourhoodOpExtra2** (HxImageData ∗src, HxImageData ∗extraIm, HxImageData ∗extraIm2, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation with extra images.

- virtual void **neighbourhoodOp** (HxImageData ∗src, HxImageData ∗kernel, **HxString** ngbName, **HxTagList** &tags)

  Neighbourhood operation with kernel.

- virtual void **queueBasedOp** (HxImageData ∗srcImg, HxImageData ∗kerImg, **HxString** genOp, **HxTagList** &tags)

  Queue based operation.

- virtual void **diyOp** (HxImageData ∗src, **HxString** diyName, **HxTagList** &tags)

  Do it yourself operation.

- virtual void **rgbOp** (**HxString** rgbName, **HxTagList** &tags)

  Display operation.

- void **MNPixOp** (HxImageData **results, int resultCnt, HxImageData **args, int argCnt, **HxString** mpoName, **HxTagList** &tags)

  M output N input pixel operation.

Geometric operations

- virtual void **geometricOp2d** (HxImageData *arg, **HxMatrix** func, **HxGeoIntType** gi, **HxVec3Double** translate, **HxValue** background)=0

  Geometric operation on 2D images.

- virtual void **restrict** (HxImageData *arg, **HxPoint** begin)
- virtual void **extend** (HxImageData *arg, **HxPoint** begin)
- virtual HxImageData * **projectDomain** (int dimension, int coordinate)=0
- virtual void **inverseProjectDomain** (int dimension, int coordinate, HxImageData *arg)=0

Sample operations

- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)=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** ()
- 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 HxImageData * **projectRange** (int dimension)
- virtual void **inverseProjectRange** (int dimension, HxImageData *arg)

Public Methods

- void **weight** (double w)
- **HxScalarDouble weight** () const

Protected Methods

- int **checkEqualImageSig** (**HxString** func, HxImageData ∗arg)
- int **checkEqualImageSizes** (**HxString** func, HxImageData ∗arg)
- int **checkEqualImageSigAndSizes** (**HxString** func, HxImageData ∗arg)
- int **checkLargerImageSigAndSizes** (**HxString** func, HxImageData ∗arg)
- int **checkProperKernelSigAndSizes** (**HxString** func, HxImageData ∗kernel, int reqDim, int reqScalar, int appliedDim=0)
- int **checkEqualImageSizesDim** (**HxString** func, 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

8.66.1 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.

8.66.2 Constructor & Destructor Documentation

8.66.2.1 HxImageData::HxImageData ()

Constructor.

```
52 {
53     _ident = _nr++;
54     _name = HxString("image") + makeString(_ident);
55     _weight = 1;
56     HxImageDataRepository::instance()->insertImage(this);
57     if (_objectObserver)
58         _objectObserver->constructed(_name);
59 }
```

8.66.2.2 HxImageData::HxImageData (const HxImageData &)

Copy constructor.

```
61                                          : HxRcObject()
62 {
63     HxEnvironment::instance()->warningStream() <<
64         "HxImageData copy constructor called" << STD_ENDL;
65     HxEnvironment::instance()->flush();
66 }
```

8.66.2.3 HxImageData::~HxImageData () [virtual]

Destructor.

```
69 {
70     HxImageDataRepository::instance()->removeImage(this);
71     if (_objectObserver)
72         _objectObserver->destructed(_name);
73 }
```

8.66.3 Member Function Documentation

8.66.3.1 int HxImageData::ident () const

Get identity.

```
77 {
78     return _ident;
79 }
```

8.66.3.2 HxString HxImageData::name () const

Get name.

```
83 {
84     return _name;
85 }
```

8.66.3.3 void HxImageData::name (HxString *s*)

Set name.

```
89 {
90     _name = s;
91 }
```

8.66.3.4 virtual int HxImageData::dimensionality () const [pure virtual]

Get dimensionality.

Reimplemented in **HxImageTem** (p. 695).

8.66.3.5 virtual int HxImageData::dimensionSize (int *i*) const [pure virtual]

Get image size in given dimension.

Reimplemented in **HxImageTem** (p. 695).

8.66.3.6 virtual HxSizes HxImageData::sizes () const [pure virtual]

Get image sizes.

Reimplemented in **HxImageTem** (p. 695).

8.66.3.7 virtual int HxImageData::numberOfPixels () const [pure virtual]

Get total number of pixels.

Reimplemented in **HxImageTem** (p. 696).

8.66.3.8 virtual int HxImageData::pixelDimensionality () const [pure virtual]

Get dimensionality of pixels.

Reimplemented in **HxImageTem** (p. 696).

8.66.3.9 virtual HxValueType HxImageData::pixelType () const [pure virtual]

Get type of pixel.

Reimplemented in **HxImageTem** (p. 696).

8.66.3.10 virtual int HxImageData::pixelPrecision () const [pure virtual]

Get pixel precision.

Reimplemented in **HxImageTem** (p. 696).

8.66.3.11 virtual HxImageSignature HxImageData::signature () const [pure virtual]

Get image signature.

Reimplemented in **HxImageTem** (p. 697).

8.66.3.12 void HxImageData::import (HxByte * *data*, HxTagList & *tags* = HxMakeTagList(), HxString *importOp* = "importPix") [virtual]

Import pixels from HxByte data.

```
1062 {
1063     inout((void*)data, HxClassName<HxByte>(), importOp, tags);
1064 }
```

8.66.3.13 void HxImageData::import (short * *data*, HxTagList & *tags* = HxMakeTagList(), HxString *importOp* = "importPix") [virtual]

Import pixels from short data.

```
1068 {
1069     inout((void*)data, HxClassName<short>(), importOp, tags);
1070 }
```

8.66.3.14 void HxImageData::import (int * *data*, HxTagList & *tags* = HxMakeTagList(), HxString *importOp* = "importPix") [virtual]

Import pixels from int data.

```
1074 {
1075     inout((void*)data, HxClassName<int>(), importOp, tags);
1076 }
```

8.66.3.15 void HxImageData::import (float * *data*, HxTagList & *tags* = HxMakeTagList(), HxString *importOp* = "importPix") [virtual]

Import pixels from float data.

```
1080 {
1081     inout((void*)data, HxClassName<float>(), importOp, tags);
1082 }
```

8.66.3.16 void HxImageData::import (double * *data*, HxTagList & *tags* = HxMakeTagList(), HxString *importOp* = "importPix") [virtual]

Import pixels from double data.

```
1086 {
1087     inout((void*)data, HxClassName<double>(), importOp, tags);
1088 }
```

8.66.3.17 void HxImageData::exportOp (HxByte * *data*, HxTagList & *tags*, HxString *exportOp* = "exportPix") [virtual]

Export pixels as HxByte data.

```
1092 {
1093     inout((void*)data, HxClassName<HxByte>(), exportOp, tags);
1094 }
```

8.66.3.18 void HxImageData::exportOp (short * *data*, HxTagList & *tags*, HxString *exportOp* = "exportPix") [virtual]

Export pixels as short data.

```
1098 {
1099     inout((void*)data, HxClassName<short>(), exportOp, tags);
1100 }
```

8.66.3.19 void HxImageData::exportOp (int * *data*, HxTagList & *tags*, HxString *exportOp* = "exportPix") [virtual]

Export pixels as int data.

```
1104 {
1105     inout((void*)data, HxClassName<int>(), exportOp, tags);
1106 }
```

8.66.3.20 void HxImageData::exportOp (float * *data*, HxTagList & *tags*, HxString *exportOp* = "exportPix") [virtual]

Export pixels as float data.

```
1110 {
1111     inout((void*)data, HxClassName<float>(), exportOp, tags);
1112 }
```

8.66.3.21 void HxImageData::exportOp (double * *data*, HxTagList & *tags*, HxString *exportOp* = "exportPix") [virtual]

Export pixels as double data.

```
1116 {
1117     inout((void*)data, HxClassName<double>(), exportOp, tags);
1118 }
```

8.66.3.22 void HxImageData::inout (void * *data*, HxString *dataType*, HxString *inOutOp*, HxTagList & *tags*) [virtual]

InOut operation.

```
1123 {
1124     inOutOp += HxString("<") + dataType + ">";
1125     HxAddTag(tags, "dataPtr", data);
1126     inout(inOutOp, tags);
1127 }
```

8.66.3.23 void HxImageData::inout (HxString *inOutOp*, HxTagList & *tags*) [virtual]

InOut operation (the real one).

```
1131 {
1132     HxImgFtorInOutKey funcKey(signature().toString(), inOutOp);
1133
1134     static HxImgFtorTableTem<HxImgFtorI1> funcTable;
1135     HxImgFtorI1* func = funcTable.find(funcKey);
1136
1137     if (func) {
1138         func->callIt(this, tags);
```

```
1139     } else {
1140         HxEnvironment::instance()->errorStream()
1141             << "Can't find " << funcKey << STD_ENDL;
1142         HxEnvironment::instance()->flush();
1143     }
1144 }
```

8.66.3.24 void HxImageData::exportExtra (HxString *exportOp*, HxImageData * *extraIm*, HxTagList & *tags*) [virtual]

Export operation with extra image.

```
1149 {
1150     HxImgFtorExportExtraKey funcKey(signature().toString(),
1151                                     extraIm->signature().toString(),
1152                                     exportOp);
1153
1154     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
1155     HxImgFtorI2* func = funcTable.find(funcKey);
1156
1157     if (func) {
1158         func->callIt(this, extraIm, tags);
1159     } else {
1160         HxEnvironment::instance()->errorStream()
1161             << "Can't find " << funcKey << STD_ENDL;
1162         HxEnvironment::instance()->flush();
1163     }
1164 }
```

8.66.3.25 void HxImageData::setPartImage (HxImageData * *src*, HxTagList & *tags*)

Set operation (the real one).

```
1213 {
1214     HxImgFtorSetKey funcKey(signature().toString(),
1215                         src->signature().toString());
1216
1217     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
1218     HxImgFtorI2* func = funcTable.find(funcKey);
1219
1220     if (func) {
1221         func->callIt(this, src, tags);
1222     } else {
1223         HxEnvironment::instance()->errorStream()
1224             << "Can't find " << funcKey << STD_ENDL;
1225         HxEnvironment::instance()->flush();
1226     }
1227 }
```

8.66.3.26 void HxImageData::setBorder (HxTagList & *tags*)

Set border operation (the real one).

```
1299 {
1300     HxImgFtorSetBorderKey funcKey(signature().toString());
```

```
1301
1302     static HxImgFtorTableTem<HxImgFtorI1> funcTable;
1303     HxImgFtorI1* func = funcTable.find(funcKey);
1304
1305     if (func) {
1306         func->callIt(this, tags);
1307     } else {
1308         HxEnvironment::instance()->errorStream()
1309             << "Can't find " << funcKey << STD_ENDL;
1310         HxEnvironment::instance()->flush();
1311     }
1312 }
```

8.66.3.27 void HxImageData::unaryPixOp (HxImageData * *srcImg*, HxString *upoName*, HxTagList & *tags*) [virtual]

Unary pixel operation.

```
354 {
355     HxImgFtorUpoKey funcKey(signature().toString(),
356                             srcImg->signature().toString(),
357                             upoName);
358
359     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
360     HxImgFtorI2* func = funcTable.find(funcKey);
361
362     if (func) {
363         func->callIt(this, srcImg, tags);
364     } else {
365         HxEnvironment::instance()->errorStream()
366             << "Can't find " << funcKey << STD_ENDL;
367         HxEnvironment::instance()->flush();
368     }
369 }
```

8.66.3.28 void HxImageData::binaryPixOp (HxImageData * *arg1*, HxImageData * *arg2*, HxString *bpoName*, HxTagList & *tags*) [virtual]

Binary pixel operation.

```
374 {
375     if (!checkEqualImageSizes("binaryPixOp", arg1) ||
376             !checkEqualImageSizes("binaryPixOp", arg2))
377         return;
378
379     HxImgFtorBpoKey funcKey(signature().toString(),
380                             arg1->signature().toString(),
381                             arg2->signature().toString(),
382                             bpoName);
383
384     static HxImgFtorTableTem<HxImgFtorI3> funcTable;
385     HxImgFtorI3* func = funcTable.find(funcKey);
386
387     if (func) {
388         func->callIt(this, arg1, arg2, tags);
389     } else {
390         HxEnvironment::instance()->errorStream()
391             << "Can't find " << funcKey << STD_ENDL;
```

```
392         HxEnvironment::instance()->flush();
393     }
394 }
```

8.66.3.29 void HxImageData::multiPixOp (HxImageData ** *srcImgs*, int *nImgs*, HxString *mpoName*, HxTagList & *tags*) [virtual]

Multi pixel operation.

```
400 {
401     HxImgFtorMpoKey funcKey(signature().toString(),
402                             srcImgs[0]->signature().toString(),
403                             mpoName);
404 
405     static HxImgFtorTableTem<HxImgFtorIM> funcTable;
406     HxImgFtorIM* func = funcTable.find(funcKey);
407 
408     if (func) {
409         func->callIt(this, srcImgs, nImgs, tags);
410     } else {
411         HxEnvironment::instance()->errorStream()
412             << "Can't find " << funcKey << STD_ENDL;
413         HxEnvironment::instance()->flush();
414     }
415 }
```

8.66.3.30 void HxImageData::MNPixOp (HxImageData ** *resImgs*, int *resCnt*, HxImageData ** *srcImgs*, int *srcCnt*, HxString *mpoName*, HxTagList & *tags*) [static]

M output N input pixel operation.

```
443 {
444     HxImgFtorMNpoKey funcKey(resImgs[0]->signature().toString(),
445                              srcImgs[0]->signature().toString(),
446                              mpoName);
447 
448     static HxImgFtorTableTem<HxImgFtorIMN> funcTable;
449     HxImgFtorIMN* 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 }
```

8.66.3.31 void HxImageData::generalizedConvolution (HxImageData * *srcImg*, 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     static HxImgFtorTableTem<HxImgFtorI3> funcTable;
486     HxImgFtorI3* func = funcTable.find(funcKey);
487
488     if (func) {
489         func->callIt(this, scratchImg, kerImg, tags);
490     } else {
491         HxEnvironment::instance()->errorStream()
492             << "Can't find " << funcKey << STD_ENDL;
493         HxEnvironment::instance()->flush();
494     }
495
496     delete scratchImg;
497 }
```

8.66.3.32 void HxImageData::generalizedConvolutionK1d (HxImageData * *srcImg*, int *dimension*, HxImageData * *kerImg*, HxString *genMul*, HxString *genAdd*, HxString *kerName*, HxTagList & *tags*) [virtual]

Generalized convolution operation in one dimension.

```
505 {
506     HxSizes borderSize;
507
508     switch (dimension) {
509     case 1  :
510         borderSize = HxSizes(kerImg->sizes().x() / 2, 0, 0);
511         break;
512     case 2  :
513         borderSize = HxSizes(0, kerImg->sizes().x() / 2, 0);
514         break;
515     case 3  :
516         borderSize = HxSizes(0, 0, kerImg->sizes().x() / 2);
517         break;
518     }
519
520     if (!checkBorderSize("generalizedConvolutionK1d", borderSize))
521         return;
522
523     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
524     HxAddTag<int>(tags, "dimension", dimension);
525
526     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
```

```
527     HxImageSignature scratchSig(kerImg->signature());
528     scratchSig.setImageDimensionality(signature().imageDimensionality());
529     HxImageData* scratchImg
530         = HxImgDataFactory::instance().makeImage(scratchSig, scratchSize);
531
532     scratchImg->setPartImage(
533                     srcImg, HxPointInt(0, 0, 0),
534                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
535     scratchImg->setBorder(borderSize, tags);
536
537     HxImgFtorGenConvK1dKey funcKey(
538                                 signature().toString(),
539                                 scratchImg->signature().toString(),
540                                 kerImg->signature().toString(),
541                                 genMul, genAdd, kerName);
542
543     static HxImgFtorTableTem<HxImgFtorI3> funcTable;
544     HxImgFtorI3* func = funcTable.find(funcKey);
545
546     if (func) {
547         func->callIt(this, scratchImg, kerImg, tags);
548     } else {
549         HxEnvironment::instance()->errorStream()
550             << "Can't find " << funcKey << STD_ENDL;
551         HxEnvironment::instance()->flush();
552     }
553
554     delete scratchImg;
555 }
```

8.66.3.33 void HxImageData::genConvSeparated (HxImageData * *srcImg*, int *dimension*, HxImageData * *kerImg1*, HxImageData * *kerImg2*, HxString *genMul*, HxString *genAdd*, HxString *kerName*, HxTagList & *tags*) [virtual]

Generalized convolution operation separated by dimension.

```
563 {
564     HxImageData* srcCopy = srcImg;
565     for (int dim=1 ; dim<=dimensionality() ; dim++) {
566         HxImageData* kerImg = (dim == dimension) ? kerImg1 : kerImg2;
567         HxSizes borderSize;
568
569         switch (dim) {
570         case 1  :
571             borderSize = HxSizes(kerImg->sizes().x() / 2, 0, 0);
572             break;
573         case 2  :
574             borderSize = HxSizes(0, kerImg->sizes().x() / 2, 0);
575             break;
576         case 3  :
577             borderSize = HxSizes(0, 0, kerImg->sizes().x() / 2);
578             break;
579         }
580
581         if (!checkBorderSize("genConvSeparated", borderSize))
582             return;
583
584         HxAddTag<HxSizes>(tags, "borderSize", borderSize);
585         HxAddTag<int>(tags, "dimension", dim);
586
587         HxSizes scratchSize = srcCopy->sizes() + borderSize * HxSizes(2, 2, 2);
```

```
588         HxImageSignature scratchSig(kerImg->signature());
589         scratchSig.setImageDimensionality(signature().imageDimensionality());
590         HxImageData* scratchImg
591             = HxImgDataFactory::instance().makeImage(scratchSig, scratchSize);
592
593         scratchImg->setPartImage(
594                         srcCopy, HxPointInt(0, 0, 0),
595                         srcCopy->sizes() - HxPointInt(1, 1, 1), borderSize);
596         scratchImg->setBorder(borderSize, tags);
597
598         HxImgFtorGenConvK1dKey funcKey(
599                                 signature().toString(),
600                                 scratchImg->signature().toString(),
601                                 kerImg->signature().toString(),
602                                 genMul, genAdd, kerName);
603
604         static HxImgFtorTableTem<HxImgFtorI3> funcTable;
605         HxImgFtorI3* func = funcTable.find(funcKey);
606
607         if (func) {
608             func->callIt(this, scratchImg, kerImg, tags);
609         } else {
610             HxEnvironment::instance()->errorStream()
611                 << "Can't find " << funcKey << STD_ENDL;
612             HxEnvironment::instance()->flush();
613         }
614
615         // setup srcCopy for the next loop (if any)
616         if (dim == 1)
617             srcCopy = HxImgDataFactory::instance().makeImage(scratchSig,
618                                                              srcImg->sizes());
619         if (dim != dimensionality())
620             srcCopy->set(this);
621         delete scratchImg;
622     } // end of dim loop
623     delete srcCopy;
624 }
```

8.66.3.34 void HxImageData::genConv2dSep (HxImageData * *srcImg*, HxImageData * *kerImg1*, HxImageData * *kerImg2*, HxString *genMul*, HxString *genAdd*, HxString *kerName*, HxTagList & *tags*) [virtual]

Separable generalized convolution operation on 2D images.

```
632 {
633     HxSizes borderSize(kerImg1->sizes().x() / 2, kerImg2->sizes().x() / 2, 0);
634
635     if (!checkBorderSize("genConv2dSep", borderSize))
636         return;
637
638     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
639
640     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
641     HxImageSignature scratchSig(kerImg1->signature());
642     scratchSig.setImageDimensionality(signature().imageDimensionality());
643     HxImageData* scratchImg
644         = HxImgDataFactory::instance().makeImage(scratchSig, scratchSize);
645
646     scratchImg->setPartImage(
647                     srcImg, HxPointInt(0, 0, 0),
648                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
```

```
649     scratchImg->setBorder(borderSize, tags);
650
651     HxImgFtorGenConv2dSepKey funcKey(
652                                 signature().toString(),
653                                 scratchImg->signature().toString(),
654                                 kerImg1->signature().toString(),
655                                 kerImg2->signature().toString(),
656                                 genMul, genAdd, kerName);
657
658     static HxImgFtorTableTem<HxImgFtorI4> funcTable;
659     HxImgFtorI4* func = funcTable.find(funcKey);
660
661     if (func) {
662         func->callIt(this, scratchImg, kerImg1, kerImg2, tags);
663     } else {
664         HxEnvironment::instance()->errorStream()
665             << "Can't find " << funcKey << STD_ENDL;
666         HxEnvironment::instance()->flush();
667     }
668
669     delete scratchImg;
670 }
```

8.66.3.35 void HxImageData::genConv3dSep (HxImageData * *srcImg*, HxImageData * *kerImg1*, HxImageData * *kerImg2*, HxImageData * *kerImg3*, HxString *genMul*, HxString *genAdd*, HxString *kerName*, HxTagList & *tags*) [virtual]

Separable generalized convolution operation on 3D images.

```
678 {
679     HxImageData* srcCopy = srcImg;
680     for (int dim=1 ; dim<=dimensionality() ; dim++) {
681         HxImageData* kerImg;
682         HxSizes borderSize;
683
684         switch (dim) {
685         case 1  :
686             kerImg = kerImg1;
687             borderSize = HxSizes(kerImg->sizes().x() / 2, 0, 0);
688             break;
689         case 2  :
690             kerImg = kerImg2;
691             borderSize = HxSizes(0, kerImg->sizes().x() / 2, 0);
692             break;
693         case 3  :
694             kerImg = kerImg3;
695             borderSize = HxSizes(0, 0, kerImg->sizes().x() / 2);
696             break;
697         }
698
699         if (!checkBorderSize("genConv3dSep", borderSize))
700             return;
701
702         HxAddTag<HxSizes>(tags, "borderSize", borderSize);
703         HxAddTag<int>(tags, "dimension", dim);
704
705         HxSizes scratchSize = srcCopy->sizes() + borderSize * HxSizes(2, 2, 2);
706         HxImageSignature scratchSig(kerImg->signature());
707         scratchSig.setImageDimensionality(signature().imageDimensionality());
708         HxImageData* scratchImg
709             = HxImgDataFactory::instance().makeImage(scratchSig, scratchSize);
```

```
710
711         scratchImg->setPartImage(
712                         srcCopy, HxPointInt(0, 0, 0),
713                         srcCopy->sizes() - HxPointInt(1, 1, 1), borderSize);
714         scratchImg->setBorder(borderSize, tags);
715
716         HxImgFtorGenConvK1dKey funcKey(
717                                 signature().toString(),
718                                 scratchImg->signature().toString(),
719                                 kerImg->signature().toString(),
720                                 genMul, genAdd, kerName);
721
722         static HxImgFtorTableTem<HxImgFtorI3> funcTable;
723         HxImgFtorI3* func = funcTable.find(funcKey);
724
725         if (func) {
726             func->callIt(this, scratchImg, kerImg, tags);
727         } else {
728             HxEnvironment::instance()->errorStream()
729                 << "Can't find " << funcKey << STD_ENDL;
730             HxEnvironment::instance()->flush();
731         }
732
733         // setup srcCopy for the next loop (if any)
734         if (dim == 1)
735             srcCopy = HxImgDataFactory::instance().makeImage(scratchSig,
736                                                              srcImg->sizes());
737         if (dim != dimensionality())
738             srcCopy->set(this);
739         delete scratchImg;
740     } // end of dim loop
741     delete srcCopy;
742 }
```

8.66.3.36 void HxImageData::recGenConv (HxImageData * *srcImg*, HxImageData * *kerImg*, HxString *genMul*, HxString *genAdd*, HxTagList & *tags*) [virtual]

Recursive generalized convolution operation.

```
749 {
750     HxSizes borderSize = kerImg->sizes() / HxSizes(2, 2, 2);
751     if (!checkBorderSize("recursiveNeighOp", borderSize))
752         return;
753
754     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
755     HxImageData* scratchImg
756         = HxImgDataFactory::instance().makeImage(
757                                       kerImg->signature(), scratchSize);
758
759     scratchImg->setPartImage(
760                     srcImg, HxPointInt(0, 0, 0),
761                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
762     scratchImg->setBorder(borderSize, tags, HXBORDERPROPAGATE);
763
764     HxImgFtorRecGenConvKey funcKey(
765                             scratchImg->signature().toString(),
766                             kerImg->signature().toString(), genMul, genAdd);
767
768     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
769     HxImgFtorI2* func = funcTable.find(funcKey);
770
771     if (func) {
```

```
772         func->callIt(scratchImg, kerImg, tags);
773     } else {
774         HxEnvironment::instance()->errorStream()
775             << "Can't find " << funcKey << STD_ENDL;
776         HxEnvironment::instance()->flush();
777     }
778
779     setPartImage(
780         scratchImg, borderSize, borderSize + sizes() - HxPointInt(1, 1, 1),
781         HxPointInt(0, 0, 0));
782
783     delete scratchImg;
784 }
```

8.66.3.37 void HxImageData::recGenConv2dSep (HxImageData * *srcImg*, HxImageData * *kerImg1*, HxImageData * *kerImg2*, HxString *genMul*, HxString *genAdd*, HxTagList & *tags*) [virtual]

Separable recursive generalized convolution operation on 2D images.

```
792 {
793     HxSizes borderSize(kerImg1->sizes().x() / 2, kerImg2->sizes().x() / 2, 0);
794     if (!checkBorderSize("recGenConv2dSep", borderSize))
795         return;
796
797     HxSizes scratchSize = srcImg->sizes() + borderSize * HxSizes(2, 2, 2);
798     HxImageData* scratchImg
799         = HxImgDataFactory::instance().makeImage(
800                                       kerImg1->signature(), scratchSize);
801
802     scratchImg->setPartImage(
803                     srcImg, HxPointInt(0, 0, 0),
804                     srcImg->sizes() - HxPointInt(1, 1, 1), borderSize);
805     scratchImg->setBorder(borderSize, tags, HXBORDERPROPAGATE);
806
807     HxImgFtorRecGenConvK1dKey funcKey(
808                             scratchImg->signature().toString(),
809                             kerImg1->signature().toString(), genMul, genAdd);
810
811     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
812     HxImgFtorI2* func = funcTable.find(funcKey);
813
814     int dimension = 1;
815     if (func) {
816         HxAddTag(tags, "borderSize", borderSize);
817         HxAddTag(tags, "dimension", dimension);
818         func->callIt(scratchImg, kerImg1, tags);
819
820         dimension = 2;
821         scratchImg->setBorder(borderSize, tags, HXBORDERPROPAGATE);
822         HxAddTag(tags, "dimension", dimension);
823         func->callIt(scratchImg, kerImg2, tags);
824     } else {
825         HxEnvironment::instance()->errorStream()
826             << "Can't find " << funcKey << STD_ENDL;
827         HxEnvironment::instance()->flush();
828     }
829
830     setPartImage(
831         scratchImg, borderSize, borderSize + sizes() - HxPointInt(1, 1, 1),
832         HxPointInt(0, 0, 0));
```

```
833
834     delete scratchImg;
835 }
```

8.66.3.38 void HxImageData::neighbourhoodOp (HxImageData * *src*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation.

```
840 {
841     HxImgFtorNgbKey funcKey(signature().toString(),
842                             src->signature().toString(), ngbName);
843
844     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
845     HxImgFtorI2* func = funcTable.find(funcKey);
846
847     if (func) {
848         func->probeOp(tags); // should put borderSize in the list
849         HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
850         HxSizes scratchSize = src->sizes() + borderSize * HxSizes(2, 2, 2);
851         HxImageData* scratch = HxImgDataFactory::instance().makeImage(
852                                             src->signature(), scratchSize);
853         scratch->setPartImage(
854                         src, HxPointInt(0, 0, 0),
855                         src->sizes() - HxPointInt(1, 1, 1), borderSize);
856         scratch->setBorder(borderSize, tags);
857
858         func->callIt(this, scratch, tags);
859
860         delete scratch;
861     } else {
862         HxEnvironment::instance()->errorStream()
863             << "Can't find " << funcKey << STD_ENDL;
864         HxEnvironment::instance()->flush();
865     }
866 }
```

8.66.3.39 void HxImageData::neighbourhoodOpExtra (HxImageData * *src*, HxImageData * *extraIm*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation with extra image.

```
871 {
872     HxImgFtorNgbExtraKey funcKey(signature().toString(),
873                                  src->signature().toString(),
874                                  extraIm->signature().toString(), ngbName);
875
876     static HxImgFtorTableTem<HxImgFtorI3> funcTable;
877     HxImgFtorI3* func = funcTable.find(funcKey);
878
879     if (func) {
880         func->probeOp(tags); // should put borderSize in the list
881         HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
882         HxSizes scratchSize = src->sizes() + borderSize * HxSizes(2, 2, 2);
883         HxImageData* scratch = HxImgDataFactory::instance().makeImage(
884                                             src->signature(), scratchSize);
885         scratch->setPartImage(
886                         src, HxPointInt(0, 0, 0),
```

```
887                         src->sizes() - HxPointInt(1, 1, 1), borderSize);
888         scratch->setBorder(borderSize, tags);
889
890         HxImageData* scratchEx = HxImgDataFactory::instance().makeImage(
891                                             extraIm->signature(), scratchSize);
892         scratchEx->setPartImage(
893                         extraIm, HxPointInt(0, 0, 0),
894                         extraIm->sizes() - HxPointInt(1, 1, 1), borderSize);
895         scratchEx->setBorder(borderSize, tags);
896
897         func->callIt(this, scratch, scratchEx, tags);
898
899         delete scratch;
900         delete scratchEx;
901     } else {
902         HxEnvironment::instance()->errorStream()
903             << "Can't find " << funcKey << STD_ENDL;
904         HxEnvironment::instance()->flush();
905     }
906 }
```

8.66.3.40 void HxImageData::neighbourhoodOpExtra2 (HxImageData * *src*, HxImageData * *extraIm*, HxImageData * *extraIm2*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation with extra images.

```
912 {
913     HxImgFtorNgbExtra2Key funcKey(signature().toString(),
914                                   src->signature().toString(),
915                                   extraIm->signature().toString(),
916                                   extraIm2->signature().toString(), ngbName);
917
918     static HxImgFtorTableTem<HxImgFtorI4> funcTable;
919     HxImgFtorI4* func = funcTable.find(funcKey);
920
921     if (func) {
922         func->probeOp(tags); // should put borderSize in the list
923         HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
924         HxSizes scratchSize = src->sizes() + borderSize * HxSizes(2, 2, 2);
925         HxImageData* scratch = HxImgDataFactory::instance().makeImage(
926                                             src->signature(), scratchSize);
927         scratch->setPartImage(
928                         src, HxPointInt(0, 0, 0),
929                         src->sizes() - HxPointInt(1, 1, 1), borderSize);
930         scratch->setBorder(borderSize, tags);
931
932         HxImageData* scratchEx = HxImgDataFactory::instance().makeImage(
933                                             extraIm->signature(), scratchSize);
934         scratchEx->setPartImage(
935                         extraIm, HxPointInt(0, 0, 0),
936                         extraIm->sizes() - HxPointInt(1, 1, 1), borderSize);
937         scratchEx->setBorder(borderSize, tags);
938
939         HxImageData* scratchEx2 = HxImgDataFactory::instance().makeImage(
940                                             extraIm2->signature(), scratchSize);
941         scratchEx2->setPartImage(
942                         extraIm2, HxPointInt(0, 0, 0),
943                         extraIm2->sizes() - HxPointInt(1, 1, 1), borderSize);
944         scratchEx2->setBorder(borderSize, tags);
945
946         func->callIt(this, scratch, scratchEx, scratchEx2, tags);
947
```

```
948         delete scratch;
949         delete scratchEx;
950         delete scratchEx2;
951     } else {
952         HxEnvironment::instance()->errorStream()
953             << "Can't find " << funcKey << STD_ENDL;
954         HxEnvironment::instance()->flush();
955     }
956 }
```

8.66.3.41 void HxImageData::neighbourhoodOp (HxImageData * *src*, HxImageData * *kernel*, HxString *ngbName*, HxTagList & *tags*) [virtual]

Neighbourhood operation with kernel.

```
962 {
963     HxImgFtorKernelNgbKey funcKey(signature().toString(),
964                                   src->signature().toString(),
965                                   kernel->signature().toString(),
966                                   ngbName);
967
968     static HxImgFtorTableTem<HxImgFtorI3> funcTable;
969     HxImgFtorI3* func = funcTable.find(funcKey);
970
971     if (func) {
972         HxAddTag(tags, "kernelSize", kernel->sizes());
973         func->probeOp(tags); // should put borderSize in the list
974         HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
975         HxSizes scratchSize = src->sizes() + borderSize * HxSizes(2, 2, 2);
976         HxImageData* scratch = HxImgDataFactory::instance().makeImage(
977                                             src->signature(), scratchSize);
978         scratch->setPartImage(
979                         src, HxPointInt(0, 0, 0),
980                         src->sizes() - HxPointInt(1, 1, 1), borderSize);
981         scratch->setBorder(borderSize, tags);
982
983         func->callIt(this, scratch, kernel, tags);
984
985         delete scratch;
986     } else {
987         HxEnvironment::instance()->errorStream()
988             << "Can't find " << funcKey << STD_ENDL;
989         HxEnvironment::instance()->flush();
990     }
991 }
```

8.66.3.42 void HxImageData::queueBasedOp (HxImageData * *srcImg*, HxImageData * *kerImg*, HxString *genOp*, HxTagList & *tags*) [virtual]

Queue based operation.

```
998                     {
999
1000     // NOT YET GET/SET PART (SCRATCH) IMAGE
1001     HxImageData * thisScratchImg=const_cast<HxImageData *>(this);
1002     HxImageData * srcScratchImg=const_cast<HxImageData *>(srcImg);
1003
1004     HxImgFtorQueueBasedKey funcKey(
```

```
1005         thisScratchImg->signature().toString(),
1006         srcScratchImg->signature().toString(),
1007         kerImg->signature().toString(),
1008         genOp);
1009
1010     typedef HxImgFtorI3 FunctorType;
1011     static HxImgFtorTableTem<FunctorType> funcTable;
1012     FunctorType* func = funcTable.find(funcKey);
1013
1014     if (func) {
1015         //HxAddTag(tags, "borderSize", borderSize);
1016         func->callIt(thisScratchImg, srcScratchImg, kerImg, tags);
1017     } else {
1018         HxEnvironment::instance()->errorStream()
1019             << "Can't find " << funcKey << STD_ENDL;
1020         HxEnvironment::instance()->flush();
1021     }
1022 }
```

8.66.3.43 void HxImageData::diyOp (HxImageData * *src*, HxString *diyName*, HxTagList & *tags*) [virtual]

Do it yourself operation.

```
1027 {
1028     HxImgFtorDiyKey funcKey(signature().toString(),
1029                             src->signature().toString(), diyName);
1030
1031     static HxImgFtorTableTem<HxImgFtorI2> funcTable;
1032     HxImgFtorI2* func = funcTable.find(funcKey);
1033
1034     if (func) {
1035         func->callIt(this, src, tags);
1036     } else {
1037         HxEnvironment::instance()->errorStream()
1038             << "Can't find " << funcKey << STD_ENDL;
1039         HxEnvironment::instance()->flush();
1040     }
1041 }
```

8.66.3.44 void HxImageData::rgbOp (HxString *rgbName*, HxTagList & *tags*) [virtual]

Display operation.

```
1045 {
1046     HxImgFtorRgbKey funcKey(signature().toString(), rgbName);
1047
1048     static HxImgFtorTableTem<HxImgFtorI1> funcTable;
1049     HxImgFtorI1* func = funcTable.find(funcKey);
1050
1051     if (func) {
1052         func->callIt(this, tags);
1053     } else {
1054         HxEnvironment::instance()->errorStream()
1055             << "Can't find " << funcKey << STD_ENDL;
1056         HxEnvironment::instance()->flush();
1057     }
1058 }
```

8.66.3.45 virtual void HxImageData::geometricOp2d (HxImageData * *arg*, HxMatrix *func*, HxGeoIntType *gi*, HxVec3Double *translate*, HxValue *background*) [pure virtual]

Geometric operation on 2D images.

Reimplemented in **HxImageTem2d** (p. 698), and **HxImageTem3d** (p. 700).

The documentation for this class was generated from the following files:

- **HxImageData.h**
- HxImageData.c

8.67 HxImageFactory Class Reference

Factory for **HxImageRep** (p. 620) 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, **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.

8.67.1 Detailed Description

Factory for **HxImageRep** (p. 620) objects.

8.67.2 Member Function Documentation

8.67.2.1 HxImageFactory & HxImageFactory::instance () [static]

The one and only instance of this class.

```
24 {
25     static HxImageFactory theFactory;
26     return theFactory;
27 }
```

8.67.2.2 HxImageRep HxImageFactory::fromSignature (const HxImageSignature & *signature*, HxSizes *sizes*)

Uninitialized image with given signature and sizes.

```
32 {
33     HxImageData* imgData = construct(signature, sizes);
34     return HxImageRep(imgData);
35 }
```

8.67.2.3 HxImageRep HxImageFactory::fromImage (const HxImageSignature & *signature*, HxImageRep *src*)

New image with given signature.

Sizes and data are taken from given image.

```
39 {
40     HxImageData* imgData = construct(signature, src.sizes());
41     if (imgData && src.pointee())
42         imgData->set(src.pointee());
43     return HxImageRep(imgData);
44 }
```

8.67.2.4 HxImageRep HxImageFactory::fromValue (const HxImageSignature & *signature*, HxSizes *sizes*, HxValue *val*)

New image with given signature and sizes.

Pixel data is initialized with given value.

```
49 {
50     HxImageData* imgData = construct(signature, sizes);
51     if (imgData)
52         imgData->set(val);
53     return HxImageRep(imgData);
54 }
```

8.67.2.5 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.

```
59 {
60     HxImageSignature signature(
61         dimensions, pixelDimensionality, INT_VALUE, sizeof(HxByte)<<3);
62     HxImageData* imgData = construct(signature, sizes);
63     if (imgData)
64         imgData->import(data);
65     return HxImageRep(imgData);
66 }
```

8.67.2.6 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.

```
71 {
72     HxImageSignature signature(
73         dimensions, pixelDimensionality, INT_VALUE, sizeof(short)<<3);
74     HxImageData* imgData = construct(signature, sizes);
75     if (imgData)
76         imgData->import(data);
77     return HxImageRep(imgData);
78 }
```

8.67.2.7 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.

```
83 {
84     HxImageSignature signature(
85         dimensions, pixelDimensionality, INT_VALUE, sizeof(int)<<3);
86     HxImageData* imgData = construct(signature, sizes);
87     if (imgData)
88         imgData->import(data);
89     return HxImageRep(imgData);
90 }
```

8.67.2.8 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.

```
95 {
96     HxImageSignature signature(
97         dimensions, pixelDimensionality, REAL_VALUE, sizeof(float)<<3);
98     HxImageData* imgData = construct(signature, sizes);
99     if (imgData)
100         imgData->import(data);
101     return HxImageRep(imgData);
102 }
```

8.67.2.9 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.

```
107 {
108     HxImageSignature signature(
109         dimensions, pixelDimensionality, REAL_VALUE, sizeof(double)<<3);
110     HxImageData* imgData = construct(signature, sizes);
111     if (imgData)
112         imgData->import(data);
113     return HxImageRep(imgData);
114 }
```

8.67.2.10 HxImageRep HxImageFactory::fromGenerator (const HxImageSignature & *signature*, const HxImageGenerator * *imgGenerator*) const

New image with given signature.

Pixel data and size are determined by image generator.

```
120 {
121     HxImageData* imgData = construct(signature, imgGenerator->domainSize());
122     HxTagList tags;
123     HxAddTag(tags, "imageGenerator", imgGenerator);
124     if (imgData)
125     {
126         imgData->inout("generate", tags);
127     }
128     return HxImageRep(imgData);
129 }
```

8.67.2.11 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.

```
135 {
136     HxImageGenerator* generator = getGenerator(generatorName);
137
138     if (!generator)
139         return HxImageRep();
```

```
140
141     generator->init(tags);
142     HxImageData* imgData = construct(signature, generator->domainSize());
143     HxTagList ttags;
144     HxAddTag(ttags, "imageGenerator", generator);
145     if (imgData)
146     {
147         imgData->inout("generate", ttags);
148     }
149     return HxImageRep(imgData);
150 }
```

8.67.2.12 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.

```
155 {
156     HxImageData* imgData = construct(signature, sizes);
157     if (imgData)
158         imgData->set(pixels);
159     return HxImageRep(imgData);
160 }
```

8.67.2.13 HxImageRep HxImageFactory::fromGrayValue (const HxImageSignature & *signature*, HxSizes *sizes*, HxByte * *pixels*)

New image with given signature and sizes.

Pixel data is initialized from given values.

```
165 {
166     HxImageData* imgData = construct(signature, sizes);
167     if (imgData)
168         imgData->set(pixels);
169     return HxImageRep(imgData);
170 }
```

8.67.2.14 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.

```
175 {
176     HxImageData* imgData = construct(signature, sizes);
177     if (imgData)
178         imgData->set(pixels);
179     return HxImageRep(imgData);
180 }
```

8.67.2.15 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.

```
186 {
187     HxImageData* imgData = construct(signature, sizes);
188     if (imgData)
189         imgData->inout(importOp, tags);
190     return HxImageRep(imgData);
191 }
```

8.67.2.16 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.

```
195 {
196     STD_OSTREAM& errorStream = HxEnvironment::instance()->errorStream();
197
198     if (!(i1 && i2)) {
199         errorStream << "Source images may not be null" << STD_ENDL;
200         HxEnvironment::instance()->flush();
201         return HxImageRep();
202     }
203     if (i1.signature() != i2.signature()) {
204         errorStream << "Source images differ in type" << STD_ENDL;
205         HxEnvironment::instance()->flush();
206         return HxImageRep();
207     }
208     if (i1.sizes() != i2.sizes()) {
209         errorStream << "Source images differ in size" << STD_ENDL;
210         HxEnvironment::instance()->flush();
211         return HxImageRep();
212     }
213     HxImageSignature signature(i1.signature());
214     HxSizes sizes(i1.sizes());
215
216     if (signature.pixelDimensionality() != 1) {
217         errorStream << "HxImageRep: images must have pixelDimensionality of 1"
218                     << STD_ENDL;
219         HxEnvironment::instance()->flush();
220         return HxImageRep();
221     }
222     signature.setPixelDimensionality(2);
223     HxImageData* imgData = construct(signature, sizes);
224     if (imgData)
225     {
226         HxTagList tags;
227         imgData->binaryPixOp(
228             i1.pointee(), i2.pointee(), "vector2d", tags);
229     }
230     return HxImageRep(imgData);
231 }
```

8.67.2.17 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.

```
235 {
236     STD_OSTREAM& errorStream = HxEnvironment::instance()->errorStream();
237     if (!(i1 && i2 && i3)) {
238         errorStream << "Source images may not be null" << STD_ENDL;
239         HxEnvironment::instance()->flush();
240         return HxImageRep();
241     }
242     if ((i1.signature() != i2.signature())
243             || (i2.signature() != i3.signature())) {
244         errorStream << "Source images differ in type" << STD_ENDL;
245         HxEnvironment::instance()->flush();
246         return HxImageRep();
247     }
248     if ((i1.sizes() != i2.sizes()) || (i2.sizes() != i3.sizes())) {
249         errorStream << "Source images differ in size" << STD_ENDL;
250         HxEnvironment::instance()->flush();
251         return HxImageRep();
252     }
253     HxImageSignature signature(i1.signature());
254     HxSizes sizes(i1.sizes());
255     if (signature.pixelDimensionality() != 1) {
256         errorStream << "HxImageRep: images must have pixelDimensionality of 1"
257                     << STD_ENDL;
258         HxEnvironment::instance()->flush();
259         return HxImageRep();
260     }
261     signature.setPixelDimensionality(3);
262     HxImageData* imgData = construct(signature, sizes);
263     if (imgData)
264     {
265         HxTagList tags;
266         HxImageData *pointees[3] = {i1.pointee(), i2.pointee(), i3.pointee()};
267         imgData->multiPixOp(pointees, 3, "vector3d", tags);
268     }
269     return HxImageRep(imgData);
270 }
```

8.67.2.18 HxImageRep HxImageFactory::fromFile (HxString *fileName*, HxTagList & *tags*)

New image read from file using HxImgFileIo lib.

```
276 {
277     HxString extName = HxSystem::instance().extName(fileName);
278
279     HxImgFileReader* reader = HxImgFileIoTable::instance().getReader(extName);
280
281     int imgNum = HxGetTag(tags, "setImage", 0);
282
283     if (!reader)
284     {
285         HxEnvironment::instance()->errorStream()
286             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
287         HxEnvironment::instance()->flush();
```

```
288         return HxImageRep();
289     }
290
291     if (reader->openFile(fileName))
292     {
293         HxImageData* imgData = 0;
294
295         bool imgIsSet = reader->setImage(imgNum);
296         reader->getInfo(tags);
297         if (imgIsSet)
298         {
299             HxPointZ begin(0, 0, 0);
300             HxTagList tags;
301             imgData = construct(
302                         reader->getSignature(), reader->getImageSize());
303             HxString inOutOp =
304                 HxString("importPix<") + reader->getDataTypeString() + ">";
305             while (reader->moreData())
306             {
307                 HxBoundingBox boundingBox(reader->getDataSize());
308                 boundingBox = boundingBox.translate(reader->getDataBegin());
309                 HxAddTag(tags, "boundingBox", boundingBox);
310                 void* data = reader->getData();
311                 if (data)
312                     imgData->inout(
313                         data, reader->getDataTypeString(), "importPix", tags);
314             }
315         }
316         else
317         {
318             HxEnvironment::instance()->errorStream()
319                 << "" << reader->errorDescription() << STD_ENDL;
320             HxEnvironment::instance()->flush();
321         }
322         reader->closeFile();
323         return imgData ? HxImageRep(imgData) : HxImageRep();
324     }
325     else
326     {
327         HxEnvironment::instance()->errorStream()
328             << "" << reader->errorDescription() << STD_ENDL;
329         HxEnvironment::instance()->flush();
330     }
331
332     return HxImageRep();
333 }
```

8.67.2.19 bool HxImageFactory::writeFile (HxImageRep *img*, HxString *fileName*, HxTagList & *tags*) const

Write image to file using HxImgFileIo lib.

```
338 {
339     if (!img) return false;
340
341     HxString extName = HxSystem::instance().extName(fileName);
342
343     HxImgFileWriter* writer = HxImgFileIoTable::instance().getWriter(extName);
344
345     if (!writer)
346     {
347         HxEnvironment::instance()->errorStream()
```

```
348             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
349         HxEnvironment::instance()->flush();
350         return false;
351     }
352
353     bool asRgb = HxGetTag(tags, "asRgb", true);
354     HxAddTag(tags, "asRgb", asRgb);
355
356     if (!(writer->openFile(fileName)            &&
357         writer->setSignature(img.signature())   &&
358         writer->setImageSize(img.sizes())       &&
359         writer->setInfo(tags)))
360     {
361         HxEnvironment::instance()->errorStream()
362             << "" << writer->errorDescription() << STD_ENDL;
363         HxEnvironment::instance()->flush();
364         return false;
365     }
366
367     HxPointZ begin(0, 0, 0);
368     HxTagList opTags;
369
370     while (writer->moreData())
371     {
372         HxString exportName = "exportPix<" + writer->getDataTypeString() + ">";
373         HxBoundingBox boundingBox(writer->getDataSize());
374         boundingBox = boundingBox.translate(writer->getDataBegin());
375         HxAddTag(opTags, "boundingBox", boundingBox);
376         void* data = writer->getData();
377         if (data)
378         {
379             HxAddTag(opTags, "dataPtr", data);
380             img.exportOp(exportName, opTags);
381             writer->putData(data);
382         }
383     }
384
385     writer->closeFile();
386
387     return true;
388 }
```

8.67.2.20 bool HxImageFactory::imagesToFile (HxImageList *imgs*, HxString *fileName*, HxTagList & *tags*) const

Write set of image to file using HxImgFileIo lib.

```
457 {
458     if (!imgs.size()) return false;
459
460     HxString extName = HxSystem::instance().extName(fileName);
461
462     HxImgFileWriter* writer = HxImgFileIoTable::instance().getWriter(extName);
463
464     if (!writer)
465     {
466         HxEnvironment::instance()->errorStream()
467             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
468         HxEnvironment::instance()->flush();
469         return false;
470     }
471
```

```
472     bool asRgb = HxGetTag(tags, "asRgb", true);
473     HxAddTag(tags, "asRgb", asRgb);
474 
475     HxImageList::const_iterator img = imgs.begin();
476 
477     if (!(writer->firstImage()                  &&
478         writer->openFile(fileName)             &&
479         writer->setSignature((*img).signature())   &&
480         writer->setImageSize((*img).sizes())       &&
481         writer->setInfo(tags)))
482     {
483         HxEnvironment::instance()->errorStream()
484             << "" << writer->errorDescription() << STD_ENDL;
485         HxEnvironment::instance()->flush();
486         return false;
487     }
488 
489     HxTagList opTags;
490 
491     while (img != imgs.end()) {
492         while (writer->moreData())
493         {
494             HxString exportName = "exportPix<" + writer->getDataTypeString() + ">";
495             HxBoundingBox boundingBox(writer->getDataSize());
496             boundingBox = boundingBox.translate(writer->getDataBegin());
497             HxAddTag(opTags, "boundingBox", boundingBox);
498             void* data = writer->getData();
499             if (data)
500             {
501                 HxAddTag(opTags, "dataPtr", data);
502                 (*img).exportOp(exportName, opTags);
503                 writer->putData(data);
504             }
505         }
506         img++;
507         if (img != imgs.end()) {
508             writer->nextImage();
509             if (!( writer->setSignature((*img).signature())   &&
510                 writer->setImageSize((*img).sizes())       &&
511                 writer->setInfo(tags)))
512             {
513                 HxEnvironment::instance()->errorStream()
514                     << "" << writer->errorDescription() << STD_ENDL;
515                 HxEnvironment::instance()->flush();
516                 return false;
517             }
518         }
519     }
520 
521     writer->closeFile();
522 
523     return true;
524 }
```

8.67.2.21 HxImageList HxImageFactory::imagesFromFile (HxString *fileName*, HxTagList & *tags*)

Read set of image from file using HxImgFileIo lib.

```
392 {
393     HxString extName = HxSystem::instance().extName(fileName);
394 
395     HxImgFileReader* reader = HxImgFileIoTable::instance().getReader(extName);
```

```
396
397     HxImageList imgs;
398
399     if (!reader)
400     {
401         HxEnvironment::instance()->errorStream()
402             << "Unknown file type \"" << extName << "\"" << STD_ENDL;
403         HxEnvironment::instance()->flush();
404         return imgs;
405     }
406
407     if (reader->openFile(fileName))
408     {
409         int imgNum;
410         for (imgNum = 0; imgNum < reader->imageCount(); imgNum++) {
411             bool imgIsSet = reader->setImage(imgNum);
412             reader->getInfo(tags);
413
414             if (imgIsSet)
415             {
416                 HxImageData* imgData = 0;
417                 HxPointZ begin(0, 0, 0);
418                 HxTagList tags;
419                 imgData = construct(
420                             reader->getSignature(), reader->getImageSize());
421                 HxString inOutOp =
422                     HxString("importPix<") + reader->getDataTypeString() + ">";
423                 while (reader->moreData())
424                 {
425                     HxBoundingBox boundingBox(reader->getDataSize());
426                     boundingBox = boundingBox.translate(reader->getDataBegin());
427                     HxAddTag(tags, "boundingBox", boundingBox);
428                     void* data = reader->getData();
429                     if (data && imgData)
430                         imgData->inout(
431                             data, reader->getDataTypeString(), "importPix", tags);
432                 }
433                 imgs += imgData ? HxImageRep(imgData) : HxImageRep();
434             }
435             else
436             {
437                 HxEnvironment::instance()->errorStream()
438                     << "" << reader->errorDescription() << STD_ENDL;
439                 HxEnvironment::instance()->flush();
440             }
441         }
442         reader->closeFile();
443     }
444     else
445     {
446         HxEnvironment::instance()->errorStream()
447             << "" << reader->errorDescription() << STD_ENDL;
448         HxEnvironment::instance()->flush();
449     }
450
451     return imgs;
452 }
```

The documentation for this class was generated from the following files:

- **HxImageFactory.h**
- HxImageFactory.c

8.68 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. 620).

- 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)

8.68.1 Detailed Description

A list of images.

8.68.2 Constructor & Destructor Documentation

8.68.2.1 HxImageList::HxImageList () [inline]

Constructor.

```
87 {
88 }
```

8.68.2.2 HxImageList::HxImageList (const HxImageList & *l*) [inline]

Copy Constructor.

```
134 {
135     _ims = l._ims;
136 }
```

8.68.2.3 HxImageList::~HxImageList () [inline]

Destructor.

```
140 {
141 }
```

8.68.3 Member Function Documentation

8.68.3.1 HxImageData** HxImageList::pointees () const

The entry points of the list.

8.68.3.2 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 }
```

8.68.3.3 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 }
```

8.68.3.4 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 }
```

8.68.3.5 HxImageRep HxImageList::multiPixOp (HxString *mpoName*, HxTagList & *tags* = HxMakeTagList()) [inline]

Apply a multiPixOp to the list, thereby contracting to an **HxImageRep** (p. 620).

```
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 }
```

8.68.3.6 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**

8.69 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.

Export operations

- void **exportOp** (**HxString** exportName, **HxTagList** &tags=HxMakeTagList()) const

  Export operation.

- void **exportOpExtra** (**HxString** exportName, HxImageRep extraIm, **HxTagList** &tags=HxMakeTagList()) const

  Export operation with an extra image.

- **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 **genConv2dSep** (HxImageRep kernel1, HxImageRep kernel2, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Generalized convolution operation on 2d images separated for each dimension.

- HxImageRep **genConv3dSep** (HxImageRep kernel1, HxImageRep kernel2, HxImageRep kernel3, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Generalized convolution operation on 3d images separated for each dimension.

Recursive generalized convolution operations.

- HxImageRep **recGenConv** (HxImageRep kerImg, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Recursive generalized convolution operation.

- HxImageRep **recGenConv2dSep** (HxImageRep kernel1, HxImageRep kernel2, **HxString** gMul, **HxString** gAdd, **ResultPrecision** resPrec=DEFAULT_PREC, **HxTagList** &tags=HxMakeTagList()) const

  Recursive generalized convolution operation on 2d images separated for each dimension.

Neighbourhood operations.

- HxImageRep **neighbourhoodOp** (**HxString** ngbName, **HxTagList** &tags=HxMakeTagList()) const

  Neighbourhood operation.

- HxImageRep **neighbourhoodOpExtra** (**HxString** ngbName, HxImageRep extraIm, **HxTagList** &tags=HxMakeTagList()) const

  Neighbourhood operation with an extra image.

- HxImageRep **neighbourhoodOpExtra2** (**HxString** ngbName, HxImageRep extraIm, HxImageRep extraIm2, **HxTagList** &tags=HxMakeTagList()) const

  Neighbourhood operation with two extra images.

- HxImageRep **neighbourhoodOp** (HxImageRep kernel, **HxString** ngbName, **HxTagList** &tags=HxMakeTagList()) const

  Neighbourhood operation with kernel.

Queue based operations.

- HxImageRep **queueBasedOp** (HxImageRep kernel, **HxString** qName, **HxTagList** &tags=HxMakeTagList()) const

  Queue based 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.

Do it yourself (DIY) operations.

- HxImageRep **diyOp** (**HxString** diyName, **HxTagList** &tags=HxMakeTagList()) const

  DIY operation.

Misc 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

  Display for 3D images.

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

  Export image data to array of int.

8.69.1 Detailed Description

Class definition of Horus image representation.

8.69.2 Member Enumeration Documentation

8.69.2.1 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.

```
239                                     {
240                                 DEFAULT_PREC, SOURCE_PREC,
241                                 ARITH_PREC, SMALL_PREC};
```

8.69.3 Constructor & Destructor Documentation

8.69.3.1 HxImageRep::HxImageRep ()

Null image.

```
126                         : _pointee(0) {
127     if (_objectObserver)
128         _objectObserver->constructed(name());
129 }
```

8.69.3.2 HxImageRep::HxImageRep (const HxImageRep & *rhs*)

Copy constructor.

```
132     : _pointee(rhs.pointee())
133 {
134     if (_objectObserver)
135         _objectObserver->constructed(name());
136 }
```

8.69.3.3 HxImageRep::~HxImageRep ()

Destructor.

```
347 {
348     if (_objectObserver)
349         _objectObserver->destructed(name());
350 }
```

8.69.4 Member Function Documentation

8.69.4.1 HxImageRep & HxImageRep::operator= (const HxImageRep & *rhs*)

Assignment operator.

```
357 {
358     _pointee = rhs._pointee;
359     return *this;
360 }
```

8.69.4.2 int HxImageRep::operator== (const HxImageRep & *arg*) const

Equality.

```
364 {
365     return ident() == arg.ident();
366 }
```

8.69.4.3 int HxImageRep::isNull () const

Indicates wether this is a valid image.

```
370 {
371     return !int(_pointee);
372 }
```

8.69.4.4 HxImageRep::operator int () const

Indicates wether this is a valid image.

```
375 {
376     return int(_pointee);
377 }
```

8.69.4.5 int HxImageRep::ident () const

The unique identifier of the image.

```
384 {
385     return pointee() ? pointee()->ident() : 0;
386 }
```

8.69.4.6 HxString HxImageRep::name () const

The (not necessarily unique) name of the image.

```
390 {
391     return pointee() ? pointee()->name() : HxString("");
392 }
```

8.69.4.7 void HxImageRep::name (HxString *s*)

Sets the name of the image.

```
396 {
397     if (pointee())
398         pointee()->name(s);
399 }
```

8.69.4.8 int HxImageRep::dimensionality () const

The number of dimensions of the image.

```
403 {
404     return pointee() ? pointee()->dimensionality() : 0;
405 }
```

8.69.4.9 int HxImageRep::dimensionSize (int *i*) const

The size of the image in the i-th dimension.

The first dimension has i = 1.

```
409 {
410     return pointee() ? pointee()->dimensionSize(i) : 0;
411 }
```

8.69.4.10 HxSizes HxImageRep::sizes () const

The dimension sizes of the image.

```
445 {
446     return pointee() ? pointee()->sizes() : HxSizes(0, 0, 0);
447 }
```

8.69.4.11 int HxImageRep::numberOfPixels () const

The total number of pixels in the image.

```
415 {
416     return pointee() ? pointee()->numberOfPixels() : 0;
417 }
```

8.69.4.12 int HxImageRep::pixelDimensionality () const

The number of dimensions of one pixel.

```
421 {
422     return pointee() ? pointee()->pixelDimensionality() : 0;
423 }
```

8.69.4.13 HxValueType HxImageRep::pixelType () const

The pixel range value type.

INT_VALUE or REAL_VALUE or COMPLEX_VALUE.

```
427 {
428     return pointee() ? pointee()->pixelType() : INT_VALUE;
429 }
```

8.69.4.14 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.

```
433 {
434     return pointee() ? pointee()->pixelPrecision() : 0;
435 }
```

8.69.4.15 HxImageSignature HxImageRep::signature () const

The signature of the image.

```
439 {
440     return pointee() ? pointee()->signature() : HxImageSignature();
441 }
```

8.69.4.16 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.

```
454 {
455     if (!pointee())
456         return HxImageRep();
457     HxMfUpo methodFrame(pointee(), upoName);
458     if (methodFrame.result())
459         methodFrame.result()->unaryPixOp(methodFrame.source(), upoName, tags);
460     return HxImageRep(methodFrame.result());
461 }
```

8.69.4.17 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.

```
486 {
487     if (!pointee())
488         return HxImageRep();
489     HxMfUpo methodFrame(pointee(), bpoName);
490     HxAddTag(tags, "value", val);
491     methodFrame.result()->unaryPixOp(methodFrame.source(), bpoName, tags);
492     return HxImageRep(methodFrame.result());
493 }
```

8.69.4.18 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.

```
498 {
499     if (!pointee())
500         return HxImageRep();
501     HxMfBpo methodFrame(pointee(), arg.pointee(), bpoName);
502     if (methodFrame.preOpIsOk())
503         methodFrame.result()->binaryPixOp(methodFrame.source1(),
504                                 methodFrame.source2(), bpoName, tags);
505     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
506                                     : HxImageRep();
507 }
```

8.69.4.19 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.

```
515 {
516     if (!pointee())
517         return HxImageRep();
518
519     HxImageData **imsPointee = new HxImageData * [args.size()+1];
520     HxImageList::const_iterator i;
521     int n = 0;
522
523     imsPointee[n++] = pointee();
524     for (i = args.begin(); i != args.end(); i++)
525         imsPointee[n++] = (*i).pointee();
526
527     HxMfMpo methodFrame(imsPointee, n, mpoName);
528
529     methodFrame.result()->multiPixOp(methodFrame.sources(),
530                             methodFrame.nSources(), mpoName, tags);
531
532     delete [] imsPointee;
533
534     return HxImageRep(methodFrame.result());
535 }
```

8.69.4.20 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.

```
543 {
544     if (!pointee())
545         return HxImageRep();
546 
547     typedef HxImageData** HxImgDataPtrArray;
548 
549     HxImageData **srcPtrs = new HxImageData * [args.size()+1];
550     HxImageList::const_iterator i;
551     int srcCnt = 0;
552 
553     srcPtrs[srcCnt++] = pointee();
554     for (i = args.begin(); i != args.end(); i++)
555         srcPtrs[srcCnt++] = (*i).pointee();
556 
557     HxMfMNpo methodFrame(srcPtrs, srcCnt, mpoName);
558 
559     delete [] srcPtrs;
560 
561     HxImageData::MNPixOp(
562         methodFrame.results(), methodFrame.resultCnt(),
563         methodFrame.sources(), methodFrame.sourceCnt(), mpoName, tags);
564 
565     HxImageList result;
566 
567     for (int n = 0; n < methodFrame.resultCnt(); n++) {
568     HxImageRep temp(methodFrame.results(n));
569         result += temp;
570     }
571 
572     return result;
573 }
```

8.69.4.21 void HxImageRep::exportOp (HxString *exportName*, HxTagList & *tags* = HxMakeTagList()) const

Export operation.

```
580 {
581     if (HxImgFtorRuleBase::instance().getIsModifying("inout", exportName))
582     {
583         HxEnvironment::instance()->errorStream()
584             << "Error: " << exportName << " is an image data modifying "
585             << "operator" << STD_ENDL;
586         return;
587     }
588     if (pointee())
589         pointee()->inout(exportName, tags);
590 }
```

8.69.4.22 void HxImageRep::exportOpExtra (HxString *exportName*, HxImageRep *extraIm*, HxTagList & *tags* = HxMakeTagList()) const

Export operation with an extra image.

```
595 {
596     HxMfExportExtra methodFrame(pointee(), extraIm.pointee(), exportName, tags);
597     if (methodFrame.preOpIsOk())
598         methodFrame.source()->exportExtra(exportName, methodFrame.extra(), tags);
599 }
```

8.69.4.23 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.

```
603 {
604     exportOp(op, tags);
605     HxValue result = HxGetTag(tags, "result", HxValue(0));
606     return result;
607 }
```

8.69.4.24 void HxImageRep::setDefaultResultPrecision (ResultPrecision *resPrec*) [static]

Set the value for DEFAULT_PREC.

```
43 {
44     _defaultResPrec = resPrec;
45 }
```

8.69.4.25 HxImageRep::ResultPrecision HxImageRep::getResultPrecision (ResultPrecision *resPrec* = DEFAULT_PREC) [static]

Get the value for the result precision.

```
49 {
50     return resPrec == DEFAULT_PREC ? _defaultResPrec : resPrec;
51 }
```

8.69.4.26 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”.

```
616 {
617     resPrec = getResultPrecision(resPrec);
618
619     HxMfGenConv methodFrame(pointee(), kernel.pointee(), resPrec);
620     if (methodFrame.preOpIsOk())
621         methodFrame.result()->generalizedConvolution(
622                                 methodFrame.source(),
623                                 methodFrame.kernel(),
624                                 gMul, gAdd, "stdKernel", tags);
625     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
626                                     : HxImageRep();
627 }
```

8.69.4.27 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 images should be two dimensional and the size in the second dimension should be 1. The convolution is performed in the specified dimension only. The precision of the result type is as specified by resPrec.

```
633 {
634     resPrec = getResultPrecision(resPrec);
635
636     HxMfGenConv methodFrame(pointee(), kernel.pointee(), resPrec, true);
637     if (methodFrame.preOpIsOk())
638         methodFrame.result()->generalizedConvolutionK1d(
639                                 methodFrame.source(),
640                                 dimension, methodFrame.kernel(),
641                                 gMul, gAdd, "stdKernel", tags);
642     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
643                                     : HxImageRep();
644 }
```

8.69.4.28 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 convolution is performed separately in all The kernel images should be two dimensional and the size in the second dimension should be 1. dimensions using the same kernel. The precision of the result type is as specified by resPrec.

```
650 {
651     if (dimensionality() == 2)
652         return genConv2dSep(kernel, kernel, gMul, gAdd, resPrec, tags);
```

```
653     return genConv3dSep(kernel, kernel, kernel, gMul, gAdd, resPrec, tags);
654 }
```

8.69.4.29 HxImageRep HxImageRep::genConv2dSep (HxImageRep *kernel1*, HxImageRep *kernel2*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Generalized convolution operation on 2d images separated for each dimension.

The result is obtained by sliding the kernels 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 first dimension and kernel2 in the second dimension. The precision of the result type is as specified by resPrec.

```
685 {
686     resPrec = getResultPrecision(resPrec);
687 
688     HxMfGenConv methodFrame(pointee(), kernel1.pointee(), kernel2.pointee(),
689                             resPrec, true);
690 
691     if (!methodFrame.preOpIsOk())
692         return HxImageRep();
693 
694     bool useK1d = HxGetTag(tags, "useK1d", false);
695 
696     if (useK1d)
697         methodFrame.result()->genConvSeparated(
698                             methodFrame.source(), 1,
699                             methodFrame.kernel(), methodFrame.kernel2(),
700                             gMul, gAdd, "stdKernel", tags);
701     else
702         methodFrame.result()->genConv2dSep(
703                             methodFrame.source(),
704                             methodFrame.kernel(), methodFrame.kernel2(),
705                             gMul, gAdd, "stdKernel", tags);
706 
707     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
708                                     : HxImageRep();
709 }
```

8.69.4.30 HxImageRep HxImageRep::genConv3dSep (HxImageRep *kernel1*, HxImageRep *kernel2*, HxImageRep *kernel3*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Generalized convolution operation on 3d images separated for each dimension.

The result is obtained by sliding the kernels 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 first dimension, etc. The precision of the result type is as specified by resPrec.

```
716 {
717     resPrec = getResultPrecision(resPrec);
718
719     HxMfGenConv methodFrame(pointee(), kernel1.pointee(), kernel2.pointee(),
720                             kernel3.pointee(), resPrec, true);
721
722     if (!methodFrame.preOpIsOk())
723         return HxImageRep();
724
725     methodFrame.result()->genConv3dSep(
726                             methodFrame.source(), methodFrame.kernel(),
727                             methodFrame.kernel2(), methodFrame.kernel3(),
728                             gMul, gAdd, "stdKernel", tags);
729
730     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
731                                     : HxImageRep();
732 }
```

8.69.4.31 HxImageRep HxImageRep::recGenConv (HxImageRep *kernel*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Recursive 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”. The operation has two passes : a forward pass (from the upper left corner to the lower right corner) and a backward pass (lower right to upper left). The forward pass uses the ”upper left” part of the kernel, the backward pass the ”lower right” part. The result at each position is written ”in the input image” so it has an influence at the next position.

```
741 {
742     resPrec = getResultPrecision(resPrec);
743
744     HxMfGenConv methodFrame(pointee(), kernel.pointee(), resPrec);
745     if (methodFrame.preOpIsOk())
746         methodFrame.result()->recGenConv(
747                                 methodFrame.source(), methodFrame.kernel(),
748                                 gMul, gAdd, tags);
749     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
750                                     : HxImageRep();
751 }
```

8.69.4.32 HxImageRep HxImageRep::recGenConv2dSep (HxImageRep *kernel1*, HxImageRep *kernel2*, HxString *gMul*, HxString *gAdd*, ResultPrecision *resPrec* = DEFAULT_PREC, HxTagList & *tags* = HxMakeTagList()) const

Recursive generalized convolution operation on 2d images 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 operation has two passes : a forward pass (from the upper left corner to the lower right corner) and a backward pass (lower right to upper left). The forward pass uses the ”upper left” part of the kernel, the backward pass the ”lower right” part. The result at each position is written ”in the input image” so it has an influence at the next position.

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 first dimension and kernel2 in the second dimension. The precision of the result type is as specified by resPrec.

```
757 {
758     resPrec = getResultPrecision(resPrec);
759
760     HxMfGenConv methodFrame(pointee(), kernel1.pointee(), kernel2.pointee(),
761                             resPrec, true);
762     if (methodFrame.preOpIsOk())
763         methodFrame.result()->recGenConv2dSep(
764                                 methodFrame.source(), methodFrame.kernel(),
765                                 methodFrame.kernel2(),
766                                 gMul, gAdd, tags);
767     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
768                                     : HxImageRep();
769 }
```

8.69.4.33 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”.

```
776 {
777     HxMfNgb methodFrame(pointee(), ngbName, tags);
778     if (methodFrame.preOpIsOk())
779         methodFrame.result()->neighbourhoodOp(
780                                 methodFrame.source(), ngbName, tags);
781     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
782                                     : HxImageRep();
783 }
```

8.69.4.34 HxImageRep HxImageRep::neighbourhoodOpExtra (HxString *ngbName*, HxImageRep *extraIm*, HxTagList & *tags* = HxMakeTagList()) const

Neighbourhood operation with an extra image.

Slides a ”neighbourhood” over the images and offers all pixels in the neighbourhood to the functor designated by ”ngbName”. The result at each position is determined by ”ngbName”.

```
788 {
789     HxMfNgb methodFrame(pointee(), extraIm.pointee(), ngbName, tags);
790     if (methodFrame.preOpIsOk())
791         methodFrame.result()->neighbourhoodOpExtra(
792                                 methodFrame.source(), methodFrame.extra(),
793                                 ngbName, tags);
794     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
795                                     : HxImageRep();
796 }
```

8.69.4.35 HxImageRep HxImageRep::neighbourhoodOpExtra2 (HxString *ngbName*, HxImageRep *extraIm*, HxImageRep *extraIm2*, HxTagList & *tags* = HxMakeTagList()) const

Neighbourhood operation with two extra images.

Slides a "neighbourhood" over the images and offers all pixels in the neighbourhood to the functor designated by "ngbName". The result at each position is determined by "ngbName".

```
801 {
802     HxMfNgb methodFrame(pointee(), extraIm.pointee(), extraIm2.pointee(),
803                         ngbName, tags);
804     if (methodFrame.preOpIsOk())
805         methodFrame.result()->neighbourhoodOpExtra2(
806                                 methodFrame.source(), methodFrame.extra(),
807                                 methodFrame.extra2(), ngbName, tags);
808     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
809                                     : HxImageRep();
810 }
```

8.69.4.36 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 as well as the corresponding pixels in the kernel to the functor designated by "ngbName". The result at each position is determined by "ngbName".

```
815 {
816     HxMfKernelNgb methodFrame(pointee(), kernel.pointee(), ngbName, tags);
817     if (methodFrame.preOpIsOk())
818         methodFrame.result()->neighbourhoodOp(
819                                 methodFrame.source(), methodFrame.kernel(),
820                                 ngbName, tags);
821     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
822                                     : HxImageRep();
823 }
```

8.69.4.37 HxImageRep HxImageRep::queueBasedOp (HxImageRep *kernel*, HxString *qName*, HxTagList & *tags* = HxMakeTagList()) const

Queue based operation.

```
831 {
832     HxMfQueueBased methodFrame(pointee(), kernel.pointee(), qName, tags);
833     if (methodFrame.preOpIsOk())
834         methodFrame.result()->queueBasedOp(
835                                 methodFrame.source(), methodFrame.kernel(),
836                                 qName, tags);
837     return methodFrame.preOpIsOk()  ? HxImageRep(methodFrame.result())
838                                     : HxImageRep();
839 }
```

8.69.4.38 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)

```
847 {
848     if ((func.nCol() != 3) || (func.nRow() != 3))
849         return errorIm("geometricOp2d: matrix needs to be 3x3");
850 
851     HxMatrix funcForw = (gt == FORWARD) ? func : func.i();
852     HxMatrix funcBack = (gt == FORWARD) ? func.i() : func;
853     if (!funcForw.valid() || !funcBack.valid())
854         return errorIm("geometricOp2d: matrix is not valid");
855 
856     HxSizes newSize = sizes();
857     HxVec3Double translate = HxVec3Double(0,0,0);
858     if (adjustSize) {
859         HxVec3Double ulP = funcForw * HxVec3Double(0,0,1);
860         HxVec3Double llP = funcForw * HxVec3Double(0, dimensionSize(2), 1);
861         HxVec3Double urP = funcForw * HxVec3Double(dimensionSize(1), 0, 1);
862         HxVec3Double lrP = funcForw * sizes();
863         ulP /= HxScalarDouble(ulP.z()); // homogeneous coordinates
864         llP /= HxScalarDouble(llP.z());
865         urP /= HxScalarDouble(urP.z());
866         lrP /= HxScalarDouble(lrP.z());
867 
868         HxVec3Double maerB = ulP.inf(llP).inf(urP).inf(lrP);
869         HxVec3Double maerE = ulP.sup(llP).sup(urP).sup(lrP);
870         newSize = HxSizes(maerE.x() - maerB.x() + 0.5,
871             maerE.y() - maerB.y() + 0.5, 1);
872         translate = HxVec3Double(maerB.x(), maerB.y(), 0);
873     }
874 
875     HxMfResize methodFrame(pointee(), newSize);
876     methodFrame.result()->geometricOp2d(methodFrame.source(), funcBack, gi,
877                                         translate, background);
878     return HxImageRep(methodFrame.result());
879 }
```

8.69.4.39 HxImageRep HxImageRep::diyOp (HxString *diyName*, HxTagList & *tags* = HxMakeTagList()) const

DIY operation.

Offers data pointers to this image and the result image to the functor designated by ”diyName”.

```
971 {
972     if (!pointee())
973         return HxImageRep();
974     HxSizes resultSizes = HxGetTag(tags, "resultSizes", sizes());
975     HxMfDiy methodFrame(pointee(), diyName, resultSizes);
976     methodFrame.result()->diyOp(methodFrame.source(), diyName, tags);
```

```
977     return HxImageRep(methodFrame.result());
978 }
```

8.69.4.40 void HxImageRep::setAt (const HxValue *v*, int *x*, int *y* = 0, int *z* = 0)

setAt is to be removed.

```
996 {
997     if (_pointee) {
998         _pointee.getUnshared();
999         _pointee->setAt(x, y, z, v);
1000     }
1001 }
```

8.69.4.41 HxValue HxImageRep::getAt (int *x*, int *y* = 0, int *z* = 0) const

Return pixel value at given position.

```
1005 {
1006     return _pointee ? _pointee->getAt(x, y, z) : HxValue(HxScalarInt(0));
1007 }
```

8.69.4.42 STD_OSTREAM & HxImageRep::printInfo (STD_OSTREAM & *os*, int *doData* = 0)

Print information.

```
1011 {
1012     return _pointee ? pointee()->printInfo(os, doData) : os ;
1013 }
```

8.69.4.43 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}

```
1035 {
1036     if (!pointee())
1037         return;
1038
1039     HxTagList tags;
1040     HxAddTag(tags, "pixels", pixels);
1041     HxAddTag(tags, "resWidth", resWidth);
1042     HxAddTag(tags, "resHeight", resHeight);
```

```
1043     HxAddTag(tags, "gi", gi);
1044
1045     if (displayMode == HxString("LogMagnitude")) {
1046         HxImageRep n2 = HxNorm2(*this);
1047         double lowVal = ((HxScalarDouble)n2.reduceOp("minAssign")).x();
1048         double highVal = ((HxScalarDouble)n2.reduceOp("maxAssign")).x();
1049         HxAddTag(tags, "lowVal", lowVal);
1050         HxAddTag(tags, "highVal", highVal);
1051
1052         n2.pointee()->rgbOp(displayMode, tags);
1053         return;
1054     }
1055
1056     if (displayMode == HxString("Stretch")) {
1057         double lowVal = ((HxVec3Double) reduceOp("minAssign")).min().x();
1058         double highVal = ((HxVec3Double) reduceOp("maxAssign")).max().x();
1059         HxAddTag(tags, "lowVal", lowVal);
1060         HxAddTag(tags, "highVal", highVal);
1061     }
1062
1063     pointee()->rgbOp(displayMode, tags);
1064 }
```

8.69.4.44 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.

```
1070 {
1071     if (!pointee())
1072         return;
1073     pointee()->getRgbPixels2d(pixels, displayMode, bufWidth, bufHeight,
1074             VX, VY, VW, VH, SX, SY, scaleX, scaleY, gi);
1075 }
```

8.69.4.45 void HxImageRep::getRgbPixels3d (int * *pixels*, HxString *displayMode*, int *dimension*, int *coordinate*, int *resWidth* = -1, int *resHeight* = -1, HxGeoIntType *gi* = NEAREST) const

Display for 3D images.

```
1080 {
1081     if (!pointee())
1082         return;
1083     HxTagList tags;
1084     HxAddTag(tags, "pixels", pixels);
1085     HxAddTag(tags, "dimension", dimension);
1086     HxAddTag(tags, "coordinate", coordinate);
1087     HxAddTag(tags, "resWidth", resWidth);
1088     HxAddTag(tags, "resHeight", resHeight);
1089     HxAddTag(tags, "gi", gi);
1090
1091     if (displayMode == HxString("LogMagnitude")) {
1092         HxImageRep n2 = HxNorm2(*this);
1093         double lowVal = ((HxScalarDouble)n2.reduceOp("minAssign")).x();
1094         double highVal = ((HxScalarDouble)n2.reduceOp("maxAssign")).x();
1095         HxAddTag(tags, "lowVal", lowVal);
1096         HxAddTag(tags, "highVal", highVal);
```

```
1097
1098         n2.pointee()->rgbOp(displayMode, tags);
1099         return;
1100     }
1101
1102     if (displayMode == HxString("Stretch")) {
1103         double lowVal = ((HxVec3Double) reduceOp("minAssign")).min().x();
1104         double highVal = ((HxVec3Double) reduceOp("maxAssign")).max().x();
1105         HxAddTag(tags, "lowVal", lowVal);
1106         HxAddTag(tags, "highVal", highVal);
1107     }
1108
1109     pointee()->rgbOp(displayMode, tags);
1110 }
```

8.69.5 Friends And Related Function Documentation

8.69.5.1 HxImageRep L_HXIMAGEREP HxProjectRange (HxImageRep *im*, int *dimension*) [friend]

Projection of the pixel range.

The function computes the projection (see **Pixels** (p. 3)) of all pixels in the input image via a unary pixel operation (see **Images** (p. 8)). Dimension starts at 1.

```
13 {
14     HxString fname("HxProjectRange");
15
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if ((im.pixelDimensionality() != 2) &&
22         (im.pixelDimensionality() != 3))
23     {
24         HxGlobalError::instance()->reportError(fname, "Operation is only valid for Vector images", HxGlobalError::HX_
25         return HxImageRep();
26     }
27     if (dimension < 1)
28     {
29         HxGlobalError::instance()->reportError(fname, "Dimension should be greater than zero", HxGlobalError::HX_GE_
30         return HxImageRep();
31     }
32     if (dimension > im.pixelDimensionality())
33     {
34         HxGlobalError::instance()->reportError(fname, "Dimension should be less than pixel dimensionality", HxGlobalE
35         return HxImageRep();
36     }
37
38     return im.projectRange(dimension);
39 }
```

8.69.5.2 HxImageRep L_HXIMAGEREP HxInverseProjectRange (HxImageRep *im*, int *dimension*, HxImageRep *arg*) [friend]

Inverse projection of the pixel range.

The function projects (see **Pixels** (p. 3)) all pixels of image im on the given dimension of image arg via a unary pixel operation (see **Images** (p. 8)). Dimension starts at 1.

```
13 {
14     HxString fname("HxInverseProjectRange");
15 
16     if (im.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (arg.isNull())
22     {
23         HxGlobalError::instance()->reportError(fname, arg.name(), "null arg image", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26 
27     if (im.dimensionality() != arg.dimensionality())
28     {
29         HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30         return HxImageRep();
31     }
32 
33     if (im.pixelDimensionality() != 1)
34     {
35         HxGlobalError::instance()->reportError(fname, "operation only valid on scalar input images", HxGlobalError::H
36         return HxImageRep();
37     }
38 
39     if (im.sizes().x() != arg.sizes().x())
40     {
41         HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
42         return HxImageRep();
43     }
44     if (im.sizes().y() != arg.sizes().y())
45     {
46         HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUAL_IMAGES);
47         return HxImageRep();
48     }
49     if (im.dimensionality() > 2)
50     {
51         if (im.sizes().z() != arg.sizes().z())
52         {
53             HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNEQUAL_IMAGES
54             return HxImageRep();
55         }
56     }
57     if (dimension < 1)
58     {
59         HxGlobalError::instance()->reportError(fname, "dimension parameter should be greater than zero", HxGlobalErro
60         return HxImageRep();
61     }
62     if (dimension > arg.pixelDimensionality())
63     {
64         HxGlobalError::instance()->reportError(fname, "dimension parameter should be less than result pixel dimensio
65         return HxImageRep();
66     }
67 
68     return im.inverseProjectRange(dimension, arg);
69 }
```

8.69.5.3 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     HxString fname("HxRestrict");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return HxImageRep();
20     }
21     if (begin.x() < 0)
22     {
23         HxGlobalError::instance()->reportError(fname, "begin.x less then 0", HxGlobalError::HX_GE_INVALID);
24         return HxImageRep();
25     }
26     if (begin.y() < 0)
27     {
28         HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
29         return HxImageRep();
30     }
31     if ((img.dimensionality() > 2) && (begin.z() < 0))
32     {
33         HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
34         return HxImageRep();
35     }
36
37     if (end.x() < begin.x())
38     {
39         HxGlobalError::instance()->reportError(fname, "end.x less than begin.x", HxGlobalError::HX_GE_INVALID);
40         return HxImageRep();
41     }
42     if (end.y() < begin.y())
43     {
44         HxGlobalError::instance()->reportError(fname, "end.y less than begin.y", HxGlobalError::HX_GE_INVALID);
45         return HxImageRep();
46     }
47     if ((img.dimensionality() > 2) && (end.z() < begin.z()))
48     {
49         HxGlobalError::instance()->reportError(fname, "end.z less than begin.z", HxGlobalError::HX_GE_INVALID);
50         return HxImageRep();
51     }
52     if (begin.x() > img.sizes().x())
53     {
54         HxGlobalError::instance()->reportError(fname, "begin.x greater then image size", HxGlobalError::HX_GE_INVALI
55         return HxImageRep();
56     }
57     if (begin.y() > img.sizes().y())
58     {
59         HxGlobalError::instance()->reportError(fname, "begin.y greater then image size", HxGlobalError::HX_GE_INVALI
60         return HxImageRep();
61     }
62     if ((img.dimensionality() > 2) && (begin.z() > img.sizes().z()))
63     {
64         HxGlobalError::instance()->reportError(fname, "begin.z greater then image size", HxGlobalError::HX_GE_INVALI
65         return HxImageRep();
66     }
67     if (end.x() > img.sizes().x())
68     {
69         HxGlobalError::instance()->reportError(fname, "end.x greater then image size", HxGlobalError::HX_GE_INVALID)
70         return HxImageRep();
71     }
72     if (end.y() > img.sizes().y())
73     {
```

```
74          HxGlobalError::instance()->reportError(fname, "end.y greater then image size", HxGlobalError::HX_GE_INVALID);
75          return HxImageRep();
76      }
77      if ((img.dimensionality() > 2) && (end.z() > img.sizes().z()))
78      {
79          HxGlobalError::instance()->reportError(fname, "end.z greater then image size", HxGlobalError::HX_GE_INVALID);
80          return HxImageRep();
81      }
82
83      return img.restrict(begin, end);
84  }
```

8.69.5.4 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      HxString fname("HxExtend");
15
16      if (img.isNull())
17      {
18          HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19          return HxImageRep();
20      }
21      if (background.isNull())
22      {
23          HxGlobalError::instance()->reportError(fname, background.name(), "null image", HxGlobalError::HX_GE_INVALID);
24          return HxImageRep();
25      }
26
27      if (img.dimensionality() != background.dimensionality())
28      {
29          HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
30          return HxImageRep();
31      }
32      if (img.pixelDimensionality() != background.pixelDimensionality())
33      {
34          HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_GE_UNEQUAL_
35          return HxImageRep();
36      }
37
38      if (begin.x() < 0)
39      {
40          HxGlobalError::instance()->reportError(fname, "begin.x less then 0", HxGlobalError::HX_GE_INVALID);
41          return HxImageRep();
42      }
43      if (begin.y() < 0)
44      {
45          HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
46          return HxImageRep();
47      }
48      if (begin.z() < 0)
49      {
50          HxGlobalError::instance()->reportError(fname, "begin.z less then 0", HxGlobalError::HX_GE_INVALID);
51          return HxImageRep();
52      }
53      if ((begin.x() + img.sizes().x()) > background.sizes().x())
54      {
```

```
55          HxGlobalError::instance()->reportError(fname, "x size of extend too large", HxGlobalError::HX_GE_INVALID);
56          return HxImageRep();
57      }
58      if ((begin.y() + img.sizes().y()) > background.sizes().y())
59      {
60          HxGlobalError::instance()->reportError(fname, "y size of extend too large", HxGlobalError::HX_GE_INVALID);
61          return HxImageRep();
62      }
63      if ((img.dimensionality() > 2) && ((begin.z() + img.sizes().z()) > background.sizes().z()))
64      {
65          HxGlobalError::instance()->reportError(fname, "z size of extend too large", HxGlobalError::HX_GE_INVALID);
66          return HxImageRep();
67      }
68
69      return img.extend(background, begin);
70  }
```

8.69.5.5 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      HxString fname("HxExtendVal");
16
17      if (img.isNull())
18      {
19          HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
20          return HxImageRep();
21      }
22
23      if (img.pixelDimensionality() == 2)
24      {
25          if ((background.tag() != HxValue::V2I) && (background.tag() != HxValue::V2D))
26          {
27              HxGlobalError::instance()->reportError(fname, "value dimensionality is not equal to pixel dimensionalitie
28              return HxImageRep();
29          }
30      }
31      if (img.pixelDimensionality() == 3)
32      {
33          if ((background.tag() != HxValue::V3I) && (background.tag() != HxValue::V3D))
34          {
35              HxGlobalError::instance()->reportError(fname, "value dimensionality is not equal to pixel dimensionalitie
36              return HxImageRep();
37          }
38      }
39
40      if (begin.x() < 0)
41      {
42          HxGlobalError::instance()->reportError(fname, "begin.x less then 0", HxGlobalError::HX_GE_INVALID);
43          return HxImageRep();
44      }
45      if (begin.y() < 0)
46      {
47          HxGlobalError::instance()->reportError(fname, "begin.y less then 0", HxGlobalError::HX_GE_INVALID);
48          return HxImageRep();
49      }
50      if (begin.z() < 0)
```

```
51     {
52         HxGlobalError::instance()->reportError(fname, "begin.z less then 0", HxGlobalError::HX_GE_INVALID);
53         return HxImageRep();
54     }
55     if ((begin.x() + img.sizes().x()) > newSize.x())
56     {
57         HxGlobalError::instance()->reportError(fname, "x size of extend too large", HxGlobalError::HX_GE_INVALID);
58         return HxImageRep();
59     }
60     if ((begin.y() + img.sizes().y()) > newSize.y())
61     {
62         HxGlobalError::instance()->reportError(fname, "y size of extend too large", HxGlobalError::HX_GE_INVALID);
63         return HxImageRep();
64     }
65     if ((begin.z() + img.sizes().z()) > newSize.z())
66     {
67         HxGlobalError::instance()->reportError(fname, "z size of extend too large", HxGlobalError::HX_GE_INVALID);
68         return HxImageRep();
69     }
70
71
72     return img.extend(newSize, background, begin);
73 }
```

8.69.5.6 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     HxString fname("HxExportMatlabPixels");
15
16     if (img.isNull())
17     {
18         HxGlobalError::instance()->reportError(fname, img.name(), "null image", HxGlobalError::HX_GE_INVALID);
19         return;
20     }
21
22     if (pixels == NULL)
23     {
24         HxGlobalError::instance()->reportError(fname, "null pointer", HxGlobalError::HX_GE_INVALID);
25         return;
26     }
27
28     img.getDoublePixels(pixels);
29 }
```

The documentation for this class was generated from the following files:

- **HxImageRep.h**
- HxImageRep.c

8.70 HxImageSeq Class Reference

Class definition for a sequence of **HxImageRep** (p. 620)'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 as **HxImageRep** (p. 620).

- void **getRgb2d** (int nr, int *pixels, **HxString** displayMode) const

  Display the specified frame in the given buffer in ARGB format using the given displayMode.

- void **getRgbPixels2d** (int nr, int *pixels, **HxString** displayMode, int resWidth=-1, int resHeight=-1, **HxGeoIntType** gi=NEAREST) const

  Display the specified frame in the given buffer in ARGB format using the given displayMode at the given resolution.

- **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

- void **setUseMDC** (int flag)

  Indicate whether the MDC library should be used to decode mpeg files (when it is available :-).

Static Public Attributes

- const int **MPEG_F** = 1
- const int **MIR_F** = 2
- const int **AVI_F** = 3

Friends

- class **HxImageSeqIter**

8.70.1 Detailed Description

Class definition for a sequence of **HxImageRep** (p. 620)'s.

Supported types:

- Mpeg video: Both Mpeg 1 and 2 are supported.
- MIR video: A simple uncompressed format. A sequence of images (**HxImageRep** (p. 620)) 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

8.70.2 Constructor & Destructor Documentation

8.70.2.1 HxImageSeq::HxImageSeq ()

Constructor.

```
31                          : _pointee(0)
32 {
33 }
```

8.70.2.2 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. 620)'s for later references. In the current implementation only contiguous frames are kept. The default is no buffering.

```
35                                                      : _pointee(0)
36 {
37 #ifndef __GNUC__
38     int pos = filename.rfind(".");
39     if (pos >=0 ) {
40         HxString ext = filename.substr(pos);
41         for (HxString::iterator i = ext.begin(); i!= ext.end(); i++)
42             *i = toupper(*i);
43         if (ext.compare(".MIR")==0)
44             _pointee = new HxImageSeqMir(filename, bufSize);
45         else {
46             HxImageSeqData* seq;
47             if (_useMDC)
48                 seq = new HxImageSeqMDC(filename, bufSize);
49             else
50                 seq = new HxImageSeqDXMedia(filename, bufSize);
51             if(seq->valid())
52                 _pointee = seq;
53             else
54                 delete seq;
55         }
56     } else {
57         HxEnvironment::instance()->errorStream()
58             << "No extension found in " << filename << STD_ENDL;
59         HxEnvironment::instance()->flush();
60     }
61 #else
62     HxImageSeqMDC* seq = new HxImageSeqMDC(filename, bufSize);
63     if(seq->valid())
64         _pointee = seq;
65     else
66         delete seq;
67 #endif
68 }
```

8.70.2.3 HxImageSeq::HxImageSeq (const HxImageSeq & *rhs*)

Copy constructor.

```
76                                          : _pointee(rhs.pointee())
77 {
78 }
```

8.70.2.4 HxImageSeq::~HxImageSeq () [virtual]

Destructor.

```
81 {
82 }
```

8.70.3 Member Function Documentation

8.70.3.1 void HxImageSeq::setUseMDC (int *flag*) [static]

Indicate whether the MDC library should be used to decode mpeg files (when it is available :-).

Default is false.

```
72 {
73     _useMDC = flag;
74 }
```

8.70.3.2 HxImageSeq & HxImageSeq::operator= (const HxImageSeq & *rhs*)

Assignment operator.

```
86 {
87     _pointee = rhs._pointee;
88     return *this;
89 }
```

8.70.3.3 int HxImageSeq::ident () const

The identity of the sequence.

```
99 {
100     return pointee() ? pointee()->ident() : 0;
101 }
```

8.70.3.4 int HxImageSeq::isNull () const

Indicates wether this is a valid sequence.

```
93 {
94     return !int(_pointee);
95 }
```

8.70.3.5 int HxImageSeq::frameWidth () const

The frame width.

```
105 {
106     return pointee() ? pointee()->frameWidth() : 0;
107 }
```

8.70.3.6 int HxImageSeq::frameHeight () const

The frame height.

```
111 {
112     return pointee() ? pointee()->frameHeight() : 0;
113 }
```

8.70.3.7 int HxImageSeq::frameDepth () const

The frame depth.

```
117 {
118     return pointee() ? pointee()->frameDepth() : 0;
119 }
```

8.70.3.8 int HxImageSeq::nrFrames () const

The number of frames.

```
123 {
124     return pointee() ? pointee()->nrFrames() : 0;
125 }
```

8.70.3.9 HxImageRep HxImageSeq::getFrame (int *nr*) const

Get the specified frame as **HxImageRep** (p. 620).

```
129 {
130     return pointee() ? pointee()->getFrame(nr) : HxImageRep();
131 }
```

8.70.3.10 void HxImageSeq::getRgb2d (int *nr*, int * *pixels*, HxString *displayMode*) const

Display the specified frame in the given buffer in ARGB format using the given displayMode.

The pixels buffer has to be allocated by the class user and should have sufficient size.

```
135 {
136     if (!pointee()) return;
137
138     pointee()->getRgb2d(nr, pixels, displayMode);
139 }
```

8.70.3.11 void HxImageSeq::getRgbPixels2d (int *nr*, int * *pixels*, HxString *displayMode*, int *resWidth* = -1, int *resHeight* = -1, HxGeoIntType *gi* = NEAREST) const

Display the specified frame in the given buffer in ARGB format using the given displayMode at the given resolution.

The pixels buffer has to be allocated by the class user and should have sufficient size.

```
144 {
145     if (!pointee()) return;
146 
147     pointee()->getRgbPixels2d(nr, pixels, displayMode, resWidth, resHeight, gi);
148 }
```

8.70.3.12 HxImageSeqIter HxImageSeq::begin ()

An iterator pointing to the first frame.

```
152 {
153     return HxImageSeqIter(this, 0);
154 }
```

8.70.3.13 HxImageSeqIter HxImageSeq::end ()

An iterator pointing beyond the last frame.

```
158 {
159     return HxImageSeqIter(this, pointee()->nrFrames());
160 }
```

8.70.3.14 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.

```
164 {
165 
166 #ifndef __GNUC__
167 
168     if (format==MIR_F) {
169         Header_Type* header = new Header_Type;
170         strcpy(header->id,"MIR");
171         header->width = frameWidth();
172         header->height = frameHeight();
173         header->depth = frameDepth();
174         header->no_of_frames = frameList.size();
175 
176         FILE *fp = fopen(filename.c_str(),"w+b");
177         if (!fp) {
178             HxEnvironment::instance()->errorStream()
179                 << "Cannot open file for writing:" << filename << STD_ENDL;
180             HxEnvironment::instance()->flush();
181             return -1;
```

```
182         }
183
184         int* buf = new int[header->width * header->height];
185         int block_size =  header->width * header->height * sizeof(int);
186
187         fwrite(header,sizeof(Header_Type),1,fp);
188
189         for (int i=0; i<frameList.size(); i++) {
190             HxImageRep im = pointee()->frame2HxImageRep(frameList[i]);
191             im.getRgbPixels2d(buf,"Direct");
192             fwrite(buf, block_size, 1, fp);
193         }
194         fclose(fp);
195
196         delete buf;
197
198         return 0;
199     }
200     else {
201         HxEnvironment::instance()->errorStream()
202             << "Writefile for this format not implemented yet" << STD_ENDL;
203         HxEnvironment::instance()->flush();
204         return -1;
205     }
206
207 #else
208
209     HxEnvironment::instance()->errorStream()
210     << "No non DX codec" << STD_ENDL;
211     HxEnvironment::instance()->flush();
212     return -1;
213
214 // __GNUC__
215 #endif
216 }
```

8.70.3.15 STD_OSTREAM & HxImageSeq::put (STD_OSTREAM & *os*) const

Put some information on the given stream.

```
220 {
221     os << "Number of Frames: " << nrFrames()
222        << ", frame sizes : " << frameWidth() << " x " << frameHeight()
223        << STD_ENDL;
224     return os;
225 }
```

The documentation for this class was generated from the following files:

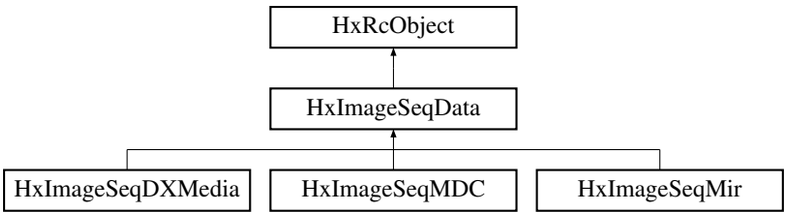
- **HxImageSeq.h**
- HxImageSeq.c

8.71 HxImageSeqData Class Reference

Base class for storage of image sequence data.

`#include <HxImageSeqData.h>`

Inheritance diagram for HxImageSeqData::



Public Methods

- **HxImageSeqData** (int bufsize)
  Constructor.
- virtual ~**HxImageSeqData** ()
  Destructor.
- int **ident** () const
  The identity of the sequence.
- virtual int **valid** ()=0
  Is this a valid object?
- virtual int **frameWidth** ()=0
  The frame width.
- virtual int **frameHeight** ()=0
  The frame height.
- virtual int **frameDepth** ()=0
  The frame depth.
- virtual int **nrFrames** ()=0
  The number of frames.
- **HxImageRep getFrame** (int fn)
  Get the specified frame.
- virtual void **getRgb2d** (int fn, int *pixels, **HxString** displayMode)
  Displays the specified frame in the buffer by calling getRgbPixels2d of the **HxImageRep** (p. 620) *corresponding to the given frame number.*
- virtual void **getRgbPixels2d** (int nr, int *pixels, **HxString** displayMode, int resWidth, int resHeight, **HxGeoIntType** gi)
  Displays the specified frame in the buffer by calling getRgbPixels2d of the **HxImageRep** (p. 620) *corresponding to the given frame number.*
- virtual **HxImageRep frame2HxImageRep** (int)=0
  "construct" an **HxImageRep** (p. 620) *for the specified frame.*

8.71.1 Detailed Description

Base class for storage of image sequence data.

The actual (pixel) data is read by specializations of this class. For that purpose, specializations have to implement the function frame2HxImageRep. The specializations also have to implement inquiry functions to provide information about image dimensions and the number of frames in the sequence.

The main purpose of this class is to provide a buffering mechanism for **HxImageRep** (p. 620)'s of the sequence as we expect processing of such a sequence will be done in a small sliding window so multiple requests for the same frame will occur.

This class also provides a default implementation

8.71.2 Constructor & Destructor Documentation

8.71.2.1 HxImageSeqData::HxImageSeqData (int *bufsize*)

Constructor.

```
21 {
22     _ident = _nr++;
23     // if == INT_MAX
24     _bufsize = bufsize;
25     _current = -1;
26     _size = 0; // nothing in the buffer
27     _buffer.clear();
28 }
```

8.71.2.2 HxImageSeqData::~HxImageSeqData () [virtual]

Destructor.

```
31 {
32     _buffer.clear();
33 }
```

8.71.3 Member Function Documentation

8.71.3.1 int HxImageSeqData::ident () const

The identity of the sequence.

```
83 {
84     return _ident;
85 }
```

8.71.3.2 virtual int HxImageSeqData::valid () [pure virtual]

Is this a valid object?

Reimplemented in **HxImageSeqDXMedia** (p. 659), and **HxImageSeqMDC** (p. 667).

8.71.3.3 virtual int HxImageSeqData::frameWidth () [pure virtual]

The frame width.

Reimplemented in **HxImageSeqDXMedia** (p. 659), and **HxImageSeqMDC** (p. 667).

8.71.3.4 virtual int HxImageSeqData::frameHeight () [pure virtual]

The frame height.

Reimplemented in **HxImageSeqDXMedia** (p. 660), and **HxImageSeqMDC** (p. 668).

8.71.3.5 virtual int HxImageSeqData::frameDepth () [pure virtual]

The frame depth.

Reimplemented in **HxImageSeqDXMedia** (p. 660), and **HxImageSeqMDC** (p. 668).

8.71.3.6 virtual int HxImageSeqData::nrFrames () [pure virtual]

The number of frames.

Reimplemented in **HxImageSeqDXMedia** (p. 660), and **HxImageSeqMDC** (p. 668).

8.71.3.7 HxImageRep HxImageSeqData::getFrame (int *fn*)

Get the specified frame.

```
37 {
38     if (_bufsize <= 0) // we are not buffering
39         return frame2HxImageRep(fn);
40
41     if (fn == _current+1) {
42         HxImageRep tmp = frame2HxImageRep(fn);
43         if (_size < _bufsize) {
44             _size++;
45         } else {
46             _buffer.pop_back();
47         }
48         _current++;
49         _buffer.push_front(tmp);
50         return tmp;
51     }
52     // check whether requested frame is in the buffer
53     if ((fn <= _current) && (fn > _current - _size)) {
54         std::list<HxImageRep>::iterator i = _buffer.begin();
55         for (int j = 0; j < _current-fn; j++, i++);
56         return *i;
57     }
58     // looks like we are "jumping" in the sequence
59     HxImageRep tmp = frame2HxImageRep(fn);
60     _current = fn;
61     _size = 1;
62     _buffer.clear();
63     _buffer.push_front(tmp);
64     return tmp;
65 }
```

8.71.3.8 void HxImageSeqData::getRgb2d (int *fn*, int * *pixels*, HxString *displayMode*) [virtual]

Displays the specified frame in the buffer by calling getRgbPixels2d of the **HxImageRep** (p. 620) corresponding to the given frame number.

Reimplemented in **HxImageSeqDXMedia** (p. 660), and **HxImageSeqMDC** (p. 668).

```
69 {
70     HxImageRep frame = getFrame(fn);
71     frame.getRgbPixels2d(pixels, displayMode);
72 }
```

8.71.3.9 void HxImageSeqData::getRgbPixels2d (int *fn*, int * *pixels*, HxString *displayMode*, int *resWidth*, int *resHeight*, HxGeoIntType *gi*) [virtual]

Displays the specified frame in the buffer by calling getRgbPixels2d of the **HxImageRep** (p. 620) corresponding to the given frame number.

Reimplemented in **HxImageSeqDXMedia** (p. 660), and **HxImageSeqMDC** (p. 668).

```
77 {
78     HxImageRep frame = getFrame(fn);
79     frame.getRgbPixels2d(pixels, displayMode, resWidth, resHeight, gi);
80 }
```

8.71.3.10 virtual HxImageRep HxImageSeqData::frame2HxImageRep (int *fn*) [pure virtual]

"construct" an **HxImageRep** (p. 620) for the specified frame.

Derived classes are to implement this function to deliver the actual image data.

Reimplemented in **HxImageSeqDXMedia** (p. 661), and **HxImageSeqMDC** (p. 669).

The documentation for this class was generated from the following files:

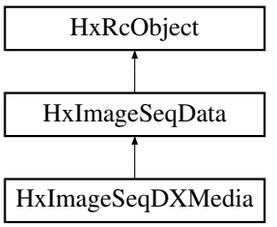
- **HxImageSeqData.h**
- HxImageSeqData.c

8.72 HxImageSeqDXMedia Class Reference

Read image sequence files using DirectMedia library.

`#include <HxImageSeqDXMedia.h>`

Inheritance diagram for HxImageSeqDXMedia::



Public Methods

- **HxImageSeqDXMedia** (**HxString** fileName, int bufSize)
  Constructor.
- virtual ~**HxImageSeqDXMedia** ()
  Destructor.
- virtual int **valid** ()
  Is this a valid object?
- virtual int **frameWidth** ()
  The frame width.
- virtual int **frameHeight** ()
  The frame height.
- virtual int **frameDepth** ()
  The frame depth.
- virtual int **nrFrames** ()
  The number of frames.
- virtual void **getRgb2d** (int fn, int *pixels, **HxString** displayMode)
  Display the specified frame in the given buffer in ARGB format using the given displayMode.
- virtual void **getRgbPixels2d** (int fn, int *pixels, **HxString** displayMode, int resWidth, int resHeight, **HxGeoIntType** gi)
  Display the specified frame in the given buffer in ARGB format using the given displayMode.
- virtual **HxImageRep frame2HxImageRep** (int)
  "construct" an **HxImageRep** (p. 620) *for the specified frame.*

8.72.1 Detailed Description

Read image sequence files using DirectMedia library.

8.72.2 Constructor & Destructor Documentation

8.72.2.1 HxImageSeqDXMedia::HxImageSeqDXMedia (HxString *fileName*, int *bufSize*)

Constructor.

```
20     : HxImageSeqData(bufSize)
21 {
22     _handle = HxDXLoadVideo(fileName.c_str());
23     if (_handle) {
24         _sizes = HxSizes(frameWidth(), frameHeight(), 1);
25     } else {
26         HxEnvironment::instance()->errorStream() <<
27             "HxImageSeqDXMedia : unable to load file " << fileName << STD_ENDL;
28         HxEnvironment::instance()->flush();
29     }
30 }
```

8.72.2.2 HxImageSeqDXMedia::~HxImageSeqDXMedia () [virtual]

Destructor.

```
39 {
40     if (_handle)
41     {
42         HxDXCloseVideo(_handle);
43     }
44 }
```

8.72.3 Member Function Documentation

8.72.3.1 int HxImageSeqDXMedia::valid () [virtual]

Is this a valid object?

Reimplemented from **HxImageSeqData** (p. 655).

```
34 {
35     return (_handle != NULL);
36 }
```

8.72.3.2 int HxImageSeqDXMedia::frameWidth () [virtual]

The frame width.

Reimplemented from **HxImageSeqData** (p. 656).

```
48 {
49     return (_handle) ? HxDXGetFrameWidth(_handle) : -1;
50 }
```

8.72.3.3 int HxImageSeqDXMedia::frameHeight () [virtual]

The frame height.

Reimplemented from **HxImageSeqData** (p. 656).

```
54 {
55     return (_handle) ? HxDXGetFrameHeight(_handle) : -1;
56 }
```

8.72.3.4 int HxImageSeqDXMedia::frameDepth () [virtual]

The frame depth.

Reimplemented from **HxImageSeqData** (p. 656).

```
60 {
61     return 1;
62 }
```

8.72.3.5 int HxImageSeqDXMedia::nrFrames () [virtual]

The number of frames.

Reimplemented from **HxImageSeqData** (p. 656).

```
66 {
67     return (_handle) ? HxDXGetLength(_handle) : 0;
68 }
```

8.72.3.6 void HxImageSeqDXMedia::getRgb2d (int *fn*, int * *pixels*, HxString *displayMode*) [virtual]

Display the specified frame in the given buffer in ARGB format using the given displayMode.

Optimized version : does not require **HxImageRep** (p. 620).

Reimplemented from **HxImageSeqData** (p. 657).

```
85 {
86     if(!_handle) return;
87
88     if(displayMode == "Direct") getRgbPixels2d(fn, pixels);
89     else
90         HxImageSeqData::getRgb2d(fn, pixels, displayMode);
91 }
```

8.72.3.7 void HxImageSeqDXMedia::getRgbPixels2d (int *fn*, int * *pixels*, HxString *displayMode*, int *resWidth*, int *resHeight*, HxGeoIntType *gi*) [virtual]

Display the specified frame in the given buffer in ARGB format using the given displayMode.

Optimized version : does not require **HxImageRep** (p. 620).

Reimplemented from **HxImageSeqData** (p. 657).

```
73 {
74     if(!_handle) return;
75
76     if((displayMode == "Direct") && (resWidth == -1) && (resHeight == -1))
77         getRgbPixels2d(fn, pixels);
78     else
79         HxImageSeqData::getRgbPixels2d(fn, pixels, displayMode,
80             resWidth, resHeight, gi);
81 }
```

8.72.3.8 HxImageRep HxImageSeqDXMedia::frame2HxImageRep (int *fn*) [virtual]

"construct" an **HxImageRep** (p. 620) for the specified frame.

Reimplemented from **HxImageSeqData** (p. 657).

```
113 {
114     HxImageSignature sig(2, 3, INT_VALUE, 8);
115     unsigned char* data = HxDXGetFrame(_handle, fn);
116     if (data) {
117         //HxTagList tags;
118         //HxAddTag(tags, "dataPtr", data);
119         //return HxImageFactory::instance().fromImport(sig, _sizes,
120         //  "importBgr", tags);
121
122         HxImageRep im = HxMakeFromByteData(3, 2, _sizes, data);
123         return im;
124     }
125     HxImageRep im2 = HxMakeFromSignature(sig, _sizes);
126     return im2;
127 }
```

The documentation for this class was generated from the following files:

- **HxImageSeqDXMedia.h**
- HxImageSeqDXMedia.c

8.73 HxImageSeqIter Class Reference

Class definition for an iterator over an **HxImageSeq** (p. 646).

`#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.

8.73.1 Detailed Description

Class definition for an iterator over an **HxImageSeq** (p. 646).

8.73.2 Constructor & Destructor Documentation

8.73.2.1 HxImageSeqIter::HxImageSeqIter ()

Constructor.

```
18                                  : _sequence(0), _framenum(0)
19 {
20 }
```

8.73.2.2 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 }
```

8.73.2.3 HxImageSeqIter::HxImageSeqIter (const HxImageSeqIter & *rhs*)

Copy constrcutor.

```
33     : _sequence(rhs._sequence), _framenum(rhs._framenum)
34 {
35 }
```

8.73.2.4 HxImageSeqIter::~HxImageSeqIter () [virtual]

Destructor.

```
38 {
39 }
```

8.73.3 Member Function Documentation

8.73.3.1 HxImageSeqIter & HxImageSeqIter::operator= (const HxImageSeqIter & *rhs*)

Assignment.

```
43 {
44      _sequence = rhs._sequence;
45      _framenum = rhs._framenum;
46      return *this;
47 }
```

8.73.3.2 HxImageSeqIter & HxImageSeqIter::operator++ ()

Increment (prefix).

```
51 {
52     _framenum++;
53     if (_framenum > _sequence->nrFrames())
54         _framenum = _sequence->nrFrames();
55     return *this;
56 }
```

8.73.3.3 HxImageSeqIter & HxImageSeqIter::operator++ (int)

Increment (postfix).

```
60 {
61     _framenum++;
62     if (_framenum > _sequence->nrFrames())
63         _framenum = _sequence->nrFrames();
64     return *this;
65 }
```

8.73.3.4 HxImageSeqIter & HxImageSeqIter::operator– ()

Decrement (prefix).

```
69 {
70     _framenum--;
71     if (_framenum < 0)
72         _framenum = 0;
73     return *this;
74 }
```

8.73.3.5 HxImageSeqIter & HxImageSeqIter::operator– (int)

Decrement (postfix).

```
78 {
79     _framenum--;
80     if (_framenum < 0)
81         _framenum = 0;
82     return *this;
83 }
```

8.73.3.6 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 }
```

8.73.3.7 HxImageRep HxImageSeqIter::operator ∗ ()

Dereferencing: get the current frame.

```
98 {
99     return _sequence->getFrame(_framenum);
100 }
```

8.73.3.8 HxImageSeqIter * HxImageSeqIter::clone () const

Make a copy.

```
104 {
105     return new HxImageSeqIter (*this);
106 }
```

8.73.3.9 bool HxImageSeqIter::operator== (const HxImageSeqIter & *x*)

Equal.

```
110 {
111     return _framenum == (x._framenum);
112 }
```

8.73.3.10 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:

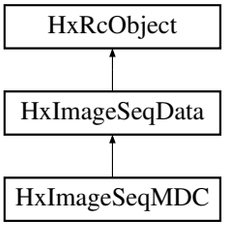
- **HxImageSeqIter.h**
- HxImageSeqIter.c

8.74 HxImageSeqMDC Class Reference

Read mpg files using MPEG Developing Classes (MDC) library.

`#include <HxImageSeqMDC.h>`

Inheritance diagram for HxImageSeqMDC::



Public Methods

- **HxImageSeqMDC** (**HxString** fileName, int bufSize)

Constructor.

- virtual ~**HxImageSeqMDC** ()

  Destructor.

- virtual int **valid** ()

  Is this a valid object?

- virtual int **frameWidth** ()

  The frame width.

- virtual int **frameHeight** ()

  The frame height.

- virtual int **frameDepth** ()

  The frame depth.

- virtual int **nrFrames** ()

  The number of frames.

- virtual void **getRgb2d** (int fn, int *pixels, **HxString** displayMode)

  Display the specified frame in the given buffer in ARGB format using the given displayMode.

- virtual void **getRgbPixels2d** (int fn, int *pixels, **HxString** displayMode, int resWidth, int resHeight, **HxGeoIntType** gi)

  Display the specified frame in the given buffer in ARGB format using the given displayMode.

- virtual **HxImageRep frame2HxImageRep** (int)

  "construct" an **HxImageRep** (p. 620) *for the specified frame.*

8.74.1 Detailed Description

Read mpg files using MPEG Developing Classes (MDC) library.

8.74.2 Constructor & Destructor Documentation

8.74.2.1 HxImageSeqMDC::HxImageSeqMDC (HxString *fileName*, int *bufSize*)

Constructor.

```
19     : HxImageSeqData(bufSize)
20 {
21     long lResult = _mpvDecoder.Initialize(fileName.c_str());
22     if (lResult == MDC_SUCCESS) {
23         if (_mpvDecoder.InitializeGenDisplayBuffer() <= 0) {
24             _valid = 0;
25         } else {
26             _mpvDecoder.SetSpeed(0);
27             _mpvDecoder.PrepareRandomAccess();
28             _mpvDecoder.JumpToEndingPicture();
```

```
29              if (_mpvDecoder.GetCurFrameNum(&lResult) == MDC_SUCCESS)
30                  _nrFrames = lResult;
31              else
32                  _nrFrames = 0;
33              WORD width = _mpvDecoder.FrameWidth();
34              WORD height = _mpvDecoder.FrameHeight();
35              _sizes = HxSizes(width, height, 1);
36              _pnOutBuf = new BYTE [width*height*3];
37              _valid = 1;
38          }
39      } else {
40          _valid = 0;
41      }
42      if (! _valid) {
43          HxEnvironment::instance()->errorStream() <<
44              "HxImageSeqMDC : unable to load file " << fileName << STD_ENDL;
45          HxEnvironment::instance()->flush();
46      }
47  }
```

8.74.2.2 HxImageSeqMDC::~HxImageSeqMDC () [virtual]

Destructor.

```
56  {
57      if (_valid) {
58          _mpvDecoder.DeleteGenDisplayBuffer();
59          _valid = false;
60      }
61  }
```

8.74.3 Member Function Documentation

8.74.3.1 int HxImageSeqMDC::valid () [virtual]

Is this a valid object?

Reimplemented from **HxImageSeqData** (p. 655).

```
51  {
52      return _valid;
53  }
```

8.74.3.2 int HxImageSeqMDC::frameWidth () [virtual]

The frame width.

Reimplemented from **HxImageSeqData** (p. 656).

```
65  {
66      return (_valid) ? _sizes.x() : -1;
67  }
```

8.74.3.3 int HxImageSeqMDC::frameHeight () [virtual]

The frame height.

Reimplemented from **HxImageSeqData** (p. 656).

```
71 {
72     return (_valid) ? _sizes.y() : -1;
73 }
```

8.74.3.4 int HxImageSeqMDC::frameDepth () [virtual]

The frame depth.

Reimplemented from **HxImageSeqData** (p. 656).

```
77 {
78     return 1;
79 }
```

8.74.3.5 int HxImageSeqMDC::nrFrames () [virtual]

The number of frames.

Reimplemented from **HxImageSeqData** (p. 656).

```
83 {
84     return _nrFrames;
85 }
```

8.74.3.6 void HxImageSeqMDC::getRgb2d (int *fn*, int * *pixels*, HxString *displayMode*) [virtual]

Display the specified frame in the given buffer in ARGB format using the given displayMode.

Optimized version : does not require **HxImageRep** (p. 620).

Reimplemented from **HxImageSeqData** (p. 657).

```
102 {
103     if(!_valid) return;
104
105     if(displayMode == "Direct") getRgbPixels2d(fn, pixels);
106     else
107         HxImageSeqData::getRgb2d(fn, pixels, displayMode);
108 }
```

8.74.3.7 void HxImageSeqMDC::getRgbPixels2d (int *fn*, int * *pixels*, HxString *displayMode*, int *resWidth*, int *resHeight*, HxGeoIntType *gi*) [virtual]

Display the specified frame in the given buffer in ARGB format using the given displayMode.

Optimized version : does not require **HxImageRep** (p. 620).

Reimplemented from **HxImageSeqData** (p. 657).

```
90 {
91     if(!_valid) return;
92
93     if((displayMode == "Direct") && (resWidth == -1) && (resHeight == -1))
94         getRgbPixels2d(fn, pixels);
95     else
96         HxImageSeqData::getRgbPixels2d(fn, pixels, displayMode,
97             resWidth, resHeight, gi);
98 }
```

8.74.3.8 HxImageRep HxImageSeqMDC::frame2HxImageRep (int *fn*) [virtual]

"construct" an **HxImageRep** (p. 620) for the specified frame.

Reimplemented from **HxImageSeqData** (p. 657).

```
136 {
137     HxImageSignature sig(2, 3, INT_VALUE, 8);
138     //unsigned char* data = HxDXGetFrame(_handle, fn);
139     WORD OutPicWidth, OutPicHeight;
140     long lItemNum = _mpvDecoder.GetBufferedFrame(_pnOutBuf, &OutPicWidth,
141                                                  &OutPicHeight, PPM, fn);
142     if (lItemNum > 0) {
143         //HxTagList tags;
144         //HxAddTag(tags, "dataPtr", data);
145         //return HxImageFactory::instance().fromImport(sig, _sizes,
146         //  "importBgr", tags);
147
148         HxImageRep im = HxMakeFromByteData(3, 2, _sizes, _pnOutBuf);
149         return im;
150     }
151     HxImageRep im2 = HxMakeFromSignature(sig, _sizes);
152     return im2;
153 }
```

The documentation for this class was generated from the following files:

- **HxImageSeqMDC.h**
- HxImageSeqMDC.c

8.75 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** ()

8.75.1 Detailed Description

Signature for a 2D image with pixels represented by an HxByte.

The documentation for this class was generated from the following file:

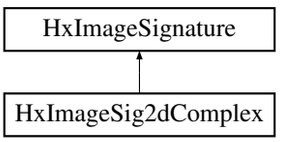
- **HxImageSig2dByte.h**

8.76 HxImageSig2dComplex Class Reference

Signature for a 2D image with pixels represented by an **HxComplex** (p. 506).

`#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** ()

8.76.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxComplex** (p. 506).

The documentation for this class was generated from the following file:

- **HxImageSig2dComplex.h**

8.77 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 **ProjectDomainImageSigType**
- typedef HxImageSig2dDouble **ArithImageSigType**
- typedef HxImageSig2dDouble **ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 1, REAL_VALUE, sizeof(PixelType) << 3) }

Public Methods

- **HxImageSig2dDouble** ()

8.77.1 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**

8.78 HxImageSig2dFloat Class Reference

Signature for a 2D image with pixels represented by a float.

`#include <HxImageSig2dFloat.h>`

Inheritance diagram for HxImageSig2dFloat::

Public Types

- typedef float **PixelType**
- typedef **HxScalarDouble ArithType**
- typedef **HxScalarDouble ArithTypeDouble**
- typedef **HxDataPtr2dScalarTem**< PixelType, ArithType > **DataPtrType**
- typedef **HxPixelAllocator**< PixelType > **Allocator**
- typedef HxImageSig2dFloat **ProjectDomainImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigType**
- typedef **HxImageSig2dDouble ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 1, REAL_VALUE, sizeof(float) << 3) }

Public Methods

- **HxImageSig2dFloat** ()

8.78.1 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**

8.79 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** ()

8.79.1 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**

8.80 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** ()

8.80.1 Detailed Description

Signature for a 2D image with pixels represented by a short.

The documentation for this class was generated from the following file:

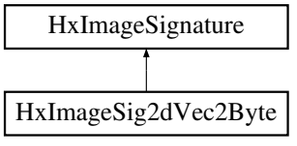
- **HxImageSig2dShort.h**

8.81 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** ()

8.81.1 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**

8.82 HxImageSig2dVec2Double Class Reference

Signature for a 2D image with pixels represented by an **HxVec2Double** (p. 1262).

`#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** ()

8.82.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec2Double** (p. 1262).

The documentation for this class was generated from the following file:

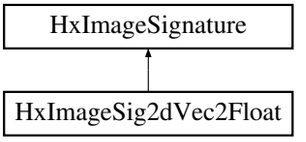
- **HxImageSig2dVec2Double.h**

8.83 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** ()

8.83.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec2Float.

The documentation for this class was generated from the following file:

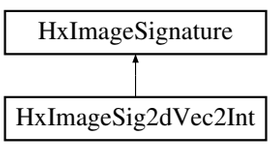
- **HxImageSig2dVec2Float.h**

8.84 HxImageSig2dVec2Int Class Reference

Signature for a 2D image with pixels represented by an **HxVec2Int** (p. 1281).

`#include <HxImageSig2dVec2Int.h>`

Inheritance diagram for HxImageSig2dVec2Int::



Public Types

- typedef **HxVec2Int PixelType**
- typedef **HxVec2Int ArithType**
- typedef **HxVec2Double ArithTypeDouble**
- typedef **HxDataPtr2dTem**< PixelType, ArithType > **DataPtrType**
- typedef **HxPixelAllocator**< PixelType > **Allocator**
- typedef HxImageSig2dVec2Int **ProjectDomainImageSigType**
- typedef HxImageSig2dVec2Int **ArithImageSigType**
- typedef **HxImageSig2dVec2Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 2, INT_VALUE, sizeof(int) << 3) }

Public Methods

- **HxImageSig2dVec2Int** ()

8.84.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec2Int** (p. 1281).

The documentation for this class was generated from the following file:

- **HxImageSig2dVec2Int.h**

8.85 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** ()

8.85.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec2Short.

The documentation for this class was generated from the following file:

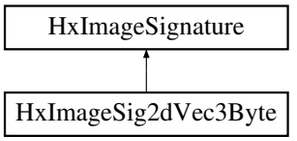
- **HxImageSig2dVec2Short.h**

8.86 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** ()

8.86.1 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**

8.87 HxImageSig2dVec3Double Class Reference

Signature for a 2D image with pixels represented by an **HxVec3Double** (p. 1301).

`#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** ()

8.87.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec3Double** (p. 1301).

The documentation for this class was generated from the following file:

- **HxImageSig2dVec3Double.h**

8.88 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** ()

8.88.1 Detailed Description

Signature for a 2D image with pixels represented by an HxVec3Float.

The documentation for this class was generated from the following file:

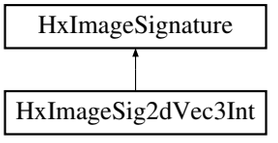
- **HxImageSig2dVec3Float.h**

8.89 HxImageSig2dVec3Int Class Reference

Signature for a 2D image with pixels represented by an **HxVec3Int** (p. 1321).

`#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** ()

8.89.1 Detailed Description

Signature for a 2D image with pixels represented by an **HxVec3Int** (p. 1321).

The documentation for this class was generated from the following file:

- **HxImageSig2dVec3Int.h**

8.90 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 **HxPixelAllocator**< PixelType > **Allocator**
- typedef HxImageSig2dVec3Short **ProjectDomainImageSigType**
- typedef **HxImageSig2dVec3Int ArithImageSigType**
- typedef **HxImageSig2dVec3Double ArithImageSigTypeDouble**
- enum { **ID** = SIG_ID(2, 3, INT_VALUE, sizeof(short) << 3) }

Public Methods

- **HxImageSig2dVec3Short** ()

8.90.1 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**

8.91 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** ()

8.91.1 Detailed Description

Signature for a 3D image with pixels represented by an HxByte.

The documentation for this class was generated from the following file:

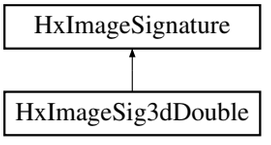
- **HxImageSig3dByte.h**

8.92 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** ()

8.92.1 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**

8.93 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** ()

8.93.1 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**

8.94 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** ()

8.94.1 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**

8.95 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** ()

8.95.1 Detailed Description

Signature for a 3D image with pixels represented by a short.

The documentation for this class was generated from the following file:

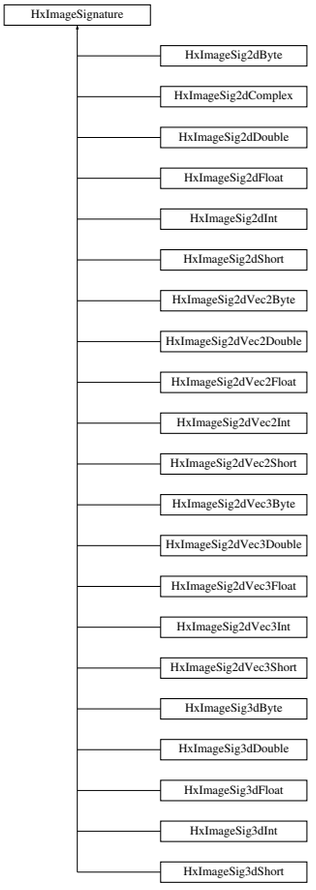
- **HxImageSig3dShort.h**

8.96 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**

8.96.1 Detailed Description

Class definition image signature.

The signature gives a description of the characteristics of an image type (class).

8.96.2 Constructor & Destructor Documentation

8.96.2.1 HxImageSignature::HxImageSignature () [inline]

Default constructor.

```
203 {
204     _imageDimensionality = 2;
205     _pixelDimensionality = 1;
206     _pixelType = INT_VALUE;
207     _pixelPrecision = sizeof(short) << 3;
208 }
```

8.96.2.2 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 }
```

8.96.2.3 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 }
```

8.96.3 Member Function Documentation

8.96.3.1 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 }
```

8.96.3.2 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 }
```

8.96.3.3 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 }
```

8.96.3.4 bool HxImageSignature::operator== (const HxImageSignature & *rhs*) const [inline]

Equal.

```
290 {
291     return isEqual(rhs);
292 }
```

8.96.3.5 int HxImageSignature::operator!= (const HxImageSignature & *rhs*) const [inline]

Not equal.

```
296 {
297     return !isEqual(rhs);
298 }
```

8.96.3.6 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 }
```

8.96.3.7 int HxImageSignature::imageDimensionality () const [inline]

The dimensionality of the image (1, 2, or 3).

```
242 {
243     return _imageDimensionality;
244 }
```

8.96.3.8 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 }
```

8.96.3.9 HxValueType HxImageSignature::pixelType () const [inline]

The type of the pixel values.

```
254 {
255     return _pixelType;
256 }
```

8.96.3.10 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 }
```

8.96.3.11 void HxImageSignature::setImageDimensionality (int *i*) [inline]

Set the image dimensionality.

```
266 {
267     _imageDimensionality = i;
268 }
```

8.96.3.12 void HxImageSignature::setPixelDimensionality (int *i*) [inline]

Set the dimensionality of the pixel values.

```
272 {
273     _pixelDimensionality = i;
274 }
```

8.96.3.13 void HxImageSignature::setPixelType (HxValueType *t*) [inline]

Set the type of the pixel values.

```
278 {
279     _pixelType = t;
280 }
```

8.96.3.14 void HxImageSignature::setPixelPrecision (int *i*) [inline]

Set the number of bits used in the representation.

```
284 {
285     _pixelPrecision = i;
286 }
```

8.96.3.15 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 }
```

8.96.3.16 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 }
```

8.96.3.17 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:

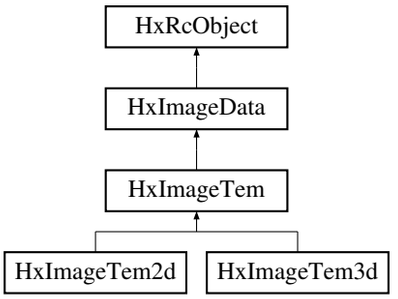
- **HxImageSignature.h**
- HxImageSignature.c

8.97 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

  Get dimensionality.

- int **dimensionSize** (int i) const

  Get image size in given dimension.

- **HxSizes sizes** () const

  Get image sizes.

- int **numberOfPixels** () const

  Get total number of pixels.

- int **pixelDimensionality** () const

  Get dimensionality of pixels.

- **HxValueType pixelType** () const

  Get type of pixel.

- int **pixelPrecision** () const

  Get pixel precision.

- **HxImageSignature signature** () const

  Get image signature.

- virtual void **set** (double ∗pixels)

- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueList-BackInserter**)=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 **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]

8.97.1 Detailed Description

template<class ImageSigT> class HxImageTem< ImageSigT >

Template class for operations that are independent of image dimensionality.

8.97.2 Member Function Documentation

8.97.2.1 template<class ImageSigT> int HxImageTem< ImageSigT >::dimensionality () const [virtual]

Get dimensionality.

Reimplemented from **HxImageData** (p. 588).

```
46 {
47     return ImageSigT().imageDimensionality();
48 }
```

8.97.2.2 template<class ImageSigT> int HxImageTem< ImageSigT >::dimensionSize (int *i*) const [virtual]

Get image size in given dimension.

Reimplemented from **HxImageData** (p. 588).

```
53 {
54     return ((i<1) || (i>dimensionality())) ? 1 : _dimSizes[i-1];
55 }
```

8.97.2.3 template<class ImageSigT> HxSizes HxImageTem< ImageSigT >::sizes () const [virtual]

Get image sizes.

Reimplemented from **HxImageData** (p. 589).

```
60 {
61     return HxSizes(_dimSizes[0], _dimSizes[1], _dimSizes[2]);
62 }
```

8.97.2.4 template<class ImageSigT> int HxImageTem< ImageSigT >::numberOfPixels () const [virtual]

Get total number of pixels.

Reimplemented from **HxImageData** (p. 589).

```
67 {
68     int dim = dimensionality();
69     int nPix = 1;
70     for (int i=1 ; i<=dim ; i++)
71         nPix *= dimensionSize(i);
72     return nPix;
73 }
```

8.97.2.5 template<class ImageSigT> int HxImageTem< ImageSigT >::pixelDimensionality () const [virtual]

Get dimensionality of pixels.

Reimplemented from **HxImageData** (p. 589).

```
78 {
79     return ImageSigT().pixelDimensionality();
80 }
```

8.97.2.6 template<class ImageSigT> HxValueType HxImageTem< ImageSigT >::pixelType () const [virtual]

Get type of pixel.

Reimplemented from **HxImageData** (p. 589).

```
85 {
86     return ImageSigT().pixelType();
87 }
```

8.97.2.7 template<class ImageSigT> int HxImageTem< ImageSigT >::pixelPrecision () const [virtual]

Get pixel precision.

Reimplemented from **HxImageData** (p. 589).

```
92 {
93     return ImageSigT().pixelPrecision();
94 }
```

8.97.2.8 template<class ImageSigT> HxImageSignature HxImageTem< ImageSigT >::signature () const [virtual]

Get image signature.

Reimplemented from **HxImageData** (p. 589).

```
99 {
100     return ImageSigT();
101 }
```

The documentation for this class was generated from the following files:

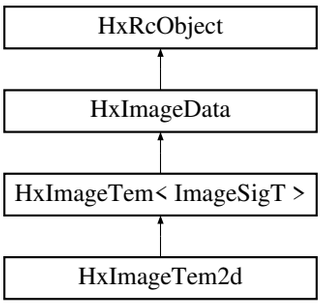
- **HxImageTem.h**
- HxImageTem.c

8.98 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 **geometricOp2d** (**HxImageData** *arg, **HxMatrix** func, **HxGeoIntType** gi, **HxVec3Double** translate, **HxValue** background)

  Geometric operation on 2D images.

- virtual **HxImageData** * **projectDomain** (int dimension, int coordinate)
- virtual void **inverseProjectDomain** (int dimension, int coordinate, **HxImageData** *arg)
- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)

- 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

8.98.1 Detailed Description

template<class ImageSigT> class HxImageTem2d< ImageSigT >

Template class for operations on 2D images.

8.98.2 Member Function Documentation

8.98.2.1 template<class ImageSigT> void HxImageTem2d< ImageSigT >::geometricOp2d (HxImageData * *arg*, HxMatrix *func*, HxGeoIntType *gi*, HxVec3Double *translate*, HxValue *background*) [virtual]

Geometric operation on 2D images.

Reimplemented from **HxImageData** (p. 605).

```
85 {
86     DataPtrType objPtr = dataPtrClone();
87     HxImageTem<ImageSigT>* argTem = (HxImageTem<ImageSigT>*) arg;
88     DataPtrType argPtr = argTem->dataPtrClone();
89     int maxArgX = (gi == LINEAR) ? arg->dimensionSize(1) - 2 : arg->dimensionSize(1) - 1;
90     int maxArgY = (gi == LINEAR) ? arg->dimensionSize(2) - 2 : arg->dimensionSize(2) - 1;
91     ArithType pixVal;
92     int x, y;
93     for (y=0; y<height(); y++) {
94         for (x=0; x<width(); x++) {
95             HxVec3Double vObj(x, y, 1);
96             HxVec3Double vArg = func * (vObj + translate);
97             double argXf = vArg.x() / vArg.z(); // homogeneous coordinates
98             double argYf = vArg.y() / vArg.z();
99             int argX = int(argXf); // truncate
100             int argY = int(argYf);
101             if ((argX < 0) || (argX > maxArgX)
102                     || (argY < 0) || (argY > maxArgY))
103                 pixVal = ArithType(background); // outside image
104             else {
105                 DataPtrType aPtr = argPtr;
106                 if (gi == LINEAR) {
107                     ArithType alpha = HxScalarDouble(argXf - argX);
108                     ArithType beta  = HxScalarDouble(argYf - argY);
109                     aPtr.incXYZ(argX, argY);
110                     ArithType v1 = aPtr.read();
111                     aPtr.incX();
112                     ArithType v2 = aPtr.read();
113                     aPtr.incXYZ(-1, 1);
114                     ArithType v3 = aPtr.read();
115                     aPtr.incX();
116                     ArithType v4 = aPtr.read();
117                     pixVal = v1 + alpha*(v2-v1) + beta*(v3-v1)
118                                 + alpha*beta*(v1-v2-v3+v4);
119                 }
120                 else { // gi == NEAREST
121                     aPtr.incXYZ(int(argXf + 0.5), int(argYf + 0.5));
122                     pixVal = aPtr.read();
123                 }
124             }
```

```
125             objPtr.write(pixVal);
126             objPtr.incX();
127         }
128     }
129 }
```

The documentation for this class was generated from the following files:

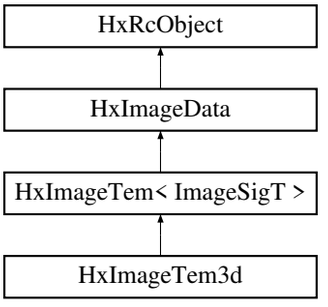
- **HxImageTem2d.h**
- HxImageTem2d.c

8.99 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 **geometricOp2d** (**HxImageData** *arg, **HxMatrix** func, **HxGeoIntType** gi, **HxVec3Double** translate, **HxValue** background)

  Geometric operation on 2D images.

- virtual **HxImageData** * **projectDomain** (int dimension, int coordinate)
- virtual void **inverseProjectDomain** (int dimension, int coordinate, **HxImageData** *arg)
- virtual void **getValues** (**HxPointListConstIter** first, **HxPointListConstIter** last, **HxValueListBackInserter**)

8.99.1 Detailed Description

template<class ImageSigT> class HxImageTem3d< ImageSigT >

Template class for operations on 3D images.

8.99.2 Member Function Documentation

8.99.2.1 template<class ImageSigT> void HxImageTem3d< ImageSigT >::geometricOp2d (HxImageData * *arg*, HxMatrix *func*, HxGeoIntType *gi*, HxVec3Double *translate*, HxValue *background*) [virtual]

Geometric operation on 2D images.

Reimplemented from **HxImageData** (p. 605).

```
94 {
95 HxEnvironment::instance()->outputStream() << "not implemented yet" << STD_ENDL;
96 HxEnvironment::instance()->flush();
97 }
```

The documentation for this class was generated from the following files:

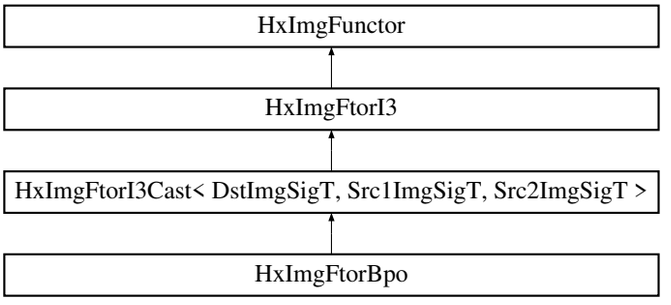
- **HxImageTem3d.h**
- HxImageTem3d.c

8.100 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** }

The type of the rule to insert in the rule base.

Public Methods

- **HxImgFtorBpo** (**RuleType** ruleType=Src1IsKey)

  Constructor.

- virtual ~**HxImgFtorBpo** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** src1Ptr, **Img3DataPtrType** src2Ptr, **HxSizes** dstSize, **HxSizes** src1Size, **HxSizes** src2Size, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Calls **HxFuncBpoDispatch** (p. 153) *to do the actual work.*

8.100.1 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.

Template parameters:

- DstImgSigT is the signature type of the destination image
- Src1ImgSigT is the signature type of the first source image
- Src2ImgSigT is the signature type of the second source image
- BpoT is the type of the pixel functor

8.100.2 Member Typedef Documentation

8.100.2.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> typedef HxImgFtorBpoKey HxImgFtorBpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 761).

8.100.3 Member Enumeration Documentation

8.100.3.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> enum HxImgFtorBpo::RuleType

The type of the rule to insert in the rule base.

- Src1IsKey implies that bpo<Src1ImgSigT,BpoT> is used as key and that the argumenttype and resulttype are inserted in the rule base
- ArgsAreKey implies that bpo<Src1ImgSigT,Src2ImgSigT,BpoT> is used as key and that the resulttype is inserted in the rule base

```
60 { Src1IsKey, ArgsAreKey };
```

8.100.4 Constructor & Destructor Documentation

8.100.4.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >::HxImgFtorBpo (RuleType *ruleType* = Src1IsKey) [inline]

Constructor.

```
25     : HxImgFtorI3Cast<DstImgSigT, Src1ImgSigT, Src2ImgSigT>(
26         HxImgFtorBpoKey(HxClassName<DstImgSigT>(), HxClassName<Src1ImgSigT>(),
27             HxClassName<Src2ImgSigT>(), HxClassName<BpoT>()))
28 {
29     switch (ruleType)
30     {
31     case Src1IsKey  :
32         HxImgFtorRuleBase::instance().setResultType(
33             HxClassName<DstImgSigT>(), "bpo",
34             HxClassName<Src1ImgSigT>(), HxClassName<BpoT>());
35         HxImgFtorRuleBase::instance().setArgumentType(
36             HxClassName<Src2ImgSigT>(), "bpo",
37             HxClassName<Src1ImgSigT>(), HxClassName<BpoT>());
38         break;
39     case ArgsAreKey :
40         HxImgFtorRuleBase::instance().setResultType(
41             HxClassName<DstImgSigT>(), "bpo",
42             HxClassName<Src1ImgSigT>(), HxClassName<Src2ImgSigT>(),
43             HxClassName<BpoT>());
44         break;
45     }
46 }
```

8.100.4.2 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >::~HxImgFtorBpo () [virtual]

Destructor.

```
50 {
51 }
```

8.100.5 Member Function Documentation

8.100.5.1 template<class DstImgSigT, class Src1ImgSigT, class Src2ImgSigT, class BpoT> void HxImgFtorBpo< DstImgSigT, Src1ImgSigT, Src2ImgSigT, BpoT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *src1Ptr*, Img3DataPtrType *src2Ptr*, HxSizes *dstSize*, HxSizes *src1Size*, HxSizes *src2Size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncBpoDispatch** (p. 153) to do the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
59 {
60     BpoT bpo(tags);
61
62     if (description) {
63         HxString v(typename BpoT::TransVarianceCategory().toString());
64         description->setVariation(v);
65     }
66
67     HxFuncBpoDispatch(dstPtr, src1Ptr, src2Ptr, dstSize, bpo);
68 }
```

The documentation for this class was generated from the following files:

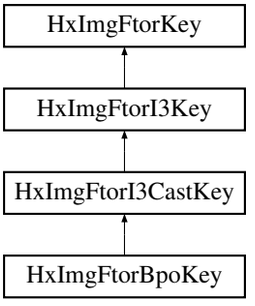
- **HxImgFtorBpo.h**
- HxImgFtorBpo.c

8.101 HxImgFtorBpoKey Class Reference

Key for **HxImgFtorBpo** (p. 700).

`#include <HxImgFtorBpoKey.h>`

Inheritance diagram for HxImgFtorBpoKey::



Public Methods

- **HxImgFtorBpoKey** (**HxString** dstImgSig, **HxString** src1ImgSig, **HxString** src2ImgSig, **HxString** bpoName)

  Constructor.

8.101.1 Detailed Description

Key for **HxImgFtorBpo** (p. 700).

8.101.2 Constructor & Destructor Documentation

8.101.2.1 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:

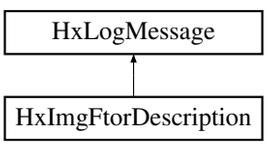
- **HxImgFtorBpoKey.h**

8.102 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

8.102.1 Detailed Description

Image functor description.

The documentation for this class was generated from the following files:

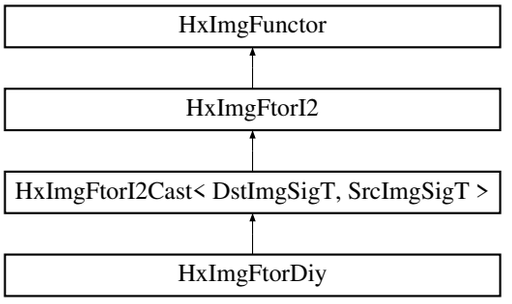
- **HxImgFtorDescription.h**
- HxImgFtorDescription.c

8.103 HxImgFtorDiy Class Template Reference

Instantiation of generic algorithm for do it yourself operations on images.

`#include <HxImgFtorDiy.h>`

Inheritance diagram for HxImgFtorDiy::



Public Types

- typedef **HxImgFtorDiyKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorDiy** ()

  Constructor.

- virtual ~**HxImgFtorDiy** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.103.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class DiyT> class HxImgFtorDiy< DstImgSigT, SrcImgSigT, DiyT >

Instantiation of generic algorithm for do it yourself operations on images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- DiyT is the type of the do it yourself functor

8.103.2 Member Typedef Documentation

8.103.2.1 template<class DstImgSigT, class SrcImgSigT, class DiyT> typedef HxImgFtorDiyKey HxImgFtorDiy::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.103.3 Constructor & Destructor Documentation

8.103.3.1 template<class DstImgSigT, class SrcImgSigT, class DiyT> HxImgFtorDiy< DstImgSigT, SrcImgSigT, DiyT >::HxImgFtorDiy () [inline]

Constructor.

```
18     : HxImgFtorI2Cast<DstImgSigT, SrcImgSigT>(
19         HxImgFtorDiyKey(
20             HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
21             HxClassName<DiyT>()))
22 {
23     HxImgFtorRuleBase::instance().setResultType(
24         HxClassName<DstImgSigT>(), "diy",
25         HxClassName<SrcImgSigT>(), HxClassName<DiyT>());
26
27 }
```

8.103.3.2 template<class DstImgSigT, class SrcImgSigT, class DiyT> HxImgFtorDiy< DstImgSigT, SrcImgSigT, DiyT >::~HxImgFtorDiy () [virtual]

Destructor.

```
31 {
32 }
```

8.103.4 Member Function Documentation

8.103.4.1 template<class DstImgSigT, class SrcImgSigT, class DiyT> void HxImgFtorDiy< DstImgSigT, SrcImgSigT, DiyT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = 0

Calls the user defined functor to do the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
41 {
42     DiyT diy(tags);
43
44     diy.doIt(dstPtr, srcPtr, dstSize, srcSize);
45 }
```

The documentation for this class was generated from the following files:

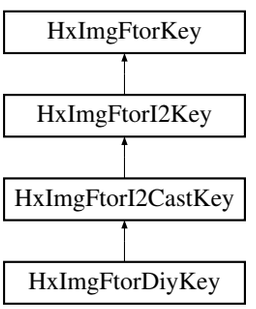
- **HxImgFtorDiy.h**
- HxImgFtorDiy.c

8.104 HxImgFtorDiyKey Class Reference

Key for **HxImgFtorDiy** (p. 705).

`#include <HxImgFtorDiyKey.h>`

Inheritance diagram for HxImgFtorDiyKey::



Public Methods

- **HxImgFtorDiyKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** diyName)

  Constructor.

8.104.1 Detailed Description

Key for **HxImgFtorDiy** (p. 705).

8.104.2 Constructor & Destructor Documentation

8.104.2.1 HxImgFtorDiyKey::HxImgFtorDiyKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *diyName*) [inline]

Constructor.

```
29     : HxImgFtorI2CastKey("HxImgFtorDiy", dstImgSig, srcImgSig)
30 {
31     addArgument(diyName);
32 }
```

The documentation for this class was generated from the following file:

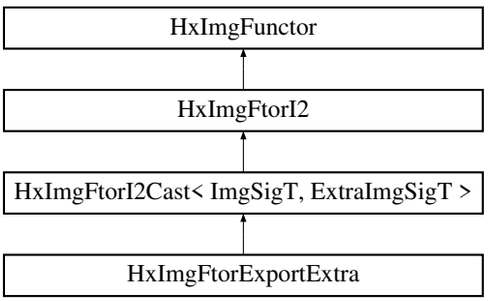
- **HxImgFtorDiyKey.h**

8.105 HxImgFtorExportExtra Class Template Reference

Instantiation of generic algorithm for export operation with an extra image.

`#include <HxImgFtorExportExtra.h>`

Inheritance diagram for HxImgFtorExportExtra::



Public Types

- typedef **HxImgFtorExportExtraKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorExportExtra** ()

  Constructor.

- virtual ~**HxImgFtorExportExtra** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** imPtr, **Img2DataPtrType** extraPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Calls **HxFuncExportExtraDispatch** (p. 156) *to dispatch the actual work.*

8.105.1 Detailed Description

template<class ImgSigT, class ExtraImgSigT, class ExportExtraT> class HxImgFtorExportExtra< ImgSigT, ExtraImgSigT, ExportExtraT >

Instantiation of generic algorithm for export operation with an extra image.

Template parameters:

- ImgSigT is the signature type of the image
- ExtraImgSigT is the signature type of the extra image
- ExportExtraT is the type of the export pixel functor

8.105.2 Member Typedef Documentation

8.105.2.1 template<class ImgSigT, class ExtraImgSigT, class ExportExtraT> typedef HxImgFtorExportExtraKey HxImgFtorExportExtra::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.105.3 Constructor & Destructor Documentation

8.105.3.1 template<class ImgSigT, class ExtraImgSigT, class ExportExtraT> HxImgFtorExportExtra< ImgSigT, ExtraImgSigT, ExportExtraT >::HxImgFtorExportExtra () [inline]

Constructor.

```
23     : HxImgFtorI2Cast<ImgSigT,ExtraImgSigT>(
24             HxImgFtorExportExtraKey(HxClassName<ImgSigT>(),
25                                     HxClassName<ExtraImgSigT>(),
26                                     HxClassName<ExportExtraT>()))
27 {
28     HxImgFtorRuleBase::instance().setExtraType(
29         HxClassName<ExtraImgSigT>(), "exportExtra",
30         HxClassName<ImgSigT>(), HxClassName<ExportExtraT>());
31 }
```

8.105.3.2 template<class ImgSigT, class ExtraImgSigT, class ExportExtraT> HxImgFtorExportExtra< ImgSigT, ExtraImgSigT, ExportExtraT >::~HxImgFtorExportExtra () [virtual]

Destructor.

```
35 {
36 }
```

8.105.4 Member Function Documentation

8.105.4.1 template<class ImgSigT, class ExtraImgSigT, class ExportExtraT> void HxImgFtorExportExtra< ImgSigT, ExtraImgSigT, ExportExtraT >::doIt (Img1DataPtrType *imPtr*, Img2DataPtrType *extraPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncExportExtraDispatch** (p. 156) to dispatch the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
43 {
44     if (description) {
45         HxString v(typename ExportExtraT::TransVarianceCategory().toString());
46         v += ", ";
47         v += typename ExportExtraT::PhaseCategory().toString();
48         description->setVariation(v);
49     }
50
51     HxBoundingBox imgBb(dstSize), regionBb(dstSize);
52     regionBb = HxGetTag(tags, "boundingBox", imgBb);
53     regionBb = regionBb.intersect(imgBb);
54     bool doOffsetExtra = HxGetTag(tags, "doOffsetExtra", true);
55
56     ExportExtraT exportOp(tags);
57
58     if (!regionBb.isEmpty()) {
59         imPtr.incXYZ(regionBb.begin().x(),
60                      regionBb.begin().y(),
61                      regionBb.begin().z());
62         if (doOffsetExtra)
63             extraPtr.incXYZ(regionBb.begin().x(),
64                             regionBb.begin().y(),
65                             regionBb.begin().z());
66         HxFuncExportExtraDispatch(imPtr, extraPtr, regionBb.size(), exportOp);
67     }
68 }
```

The documentation for this class was generated from the following files:

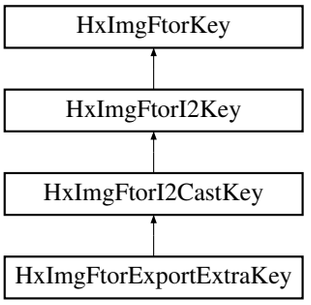
- **HxImgFtorExportExtra.h**
- HxImgFtorExportExtra.c

8.106 HxImgFtorExportExtraKey Class Reference

Key for **HxImgFtorExportExtra** (p. 708).

`#include <HxImgFtorExportExtraKey.h>`

Inheritance diagram for HxImgFtorExportExtraKey::



Public Methods

- **HxImgFtorExportExtraKey** (**HxString** imgSig, **HxString** extraImSig, **HxString** exportName)
  Constructor.

8.106.1 Detailed Description

Key for **HxImgFtorExportExtra** (p. 708).

8.106.2 Constructor & Destructor Documentation

8.106.2.1 HxImgFtorExportExtraKey::HxImgFtorExportExtraKey (HxString *imgSig*, HxString *extraImSig*, HxString *exportName*) [inline]

Constructor.

```
30     : HxImgFtorI2CastKey("HxImgFtorExportExtra", imgSig, extraImSig)
31 {
32     addArgument(exportName);
33 }
```

The documentation for this class was generated from the following file:

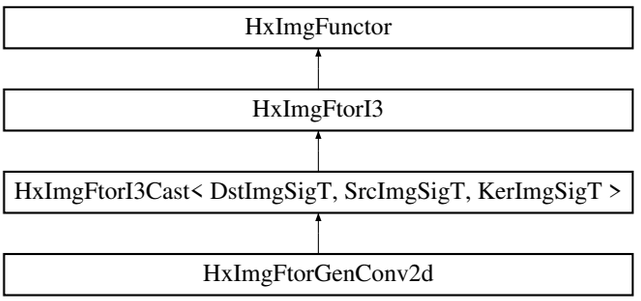
- **HxImgFtorExportExtraKey.h**

8.107 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.

Public Methods

- **HxImgFtorGenConv2d** ()

  Constructor.

- virtual ~**HxImgFtorGenConv2d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.107.1 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.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

8.107.2 Member Typedef Documentation

8.107.2.1 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. 761).

8.107.3 Constructor & Destructor Documentation

8.107.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv2d () [inline]

Constructor.

```
33         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
34             HxImgFtorGenConvKey(HxClassName<DstImgSigT>(),
35                 HxClassName<SrcImgSigT>(), HxClassName<KerImgSigT>(),
36                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
37                 HxClassName<KernelT>()))
38 {
39 #ifdef CD_TRACE
40     HxEnvironment::instance()->outputStream()
41         << "HxImgFtorGenConv2d::HxImgFtorGenConv2d()" << STD_ENDL;
42 #endif
43 }
```

8.107.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv2d () [virtual]

Destructor.

```
50 {
51 #ifdef CD_TRACE
52     HxEnvironment::instance()->outputStream()
53         << "HxImgFtorGenConv2d::~HxImgFtorGenConv2d()" << STD_ENDL;
54 #endif
55 }
```

8.107.4 Member Function Documentation

8.107.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv2d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = kerSize/2

kerPtr Input image, IS = kerSize, IBS = 0

Calls **HxFuncGenConv2dDispatch** (p. 157) to dispatch the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
66 {
67     bool rowpixfunc = HxGetTag(tags, "rowpixfunc", false);
68
69     if (rowpixfunc && description)
70         description->setVariation("rowpixfunc");
71
72     PixOpT pixOp(tags);
73     RedOpT redOp(tags);
74     KernelT kernel(kerPtr, kerSize, tags);
75
76     HxFuncGenConv2dDispatch(dstPtr, srcPtr, kernel, dstSize,
77                             pixOp, redOp, rowpixfunc);
78 }
```

The documentation for this class was generated from the following files:

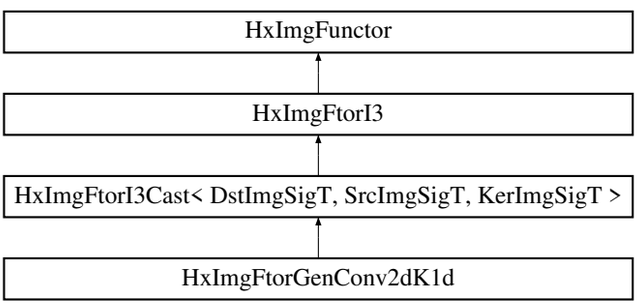
- **HxImgFtorGenConv2d.h**
- HxImgFtorGenConv2d.c

8.108 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.

Public Methods

- **HxImgFtorGenConv2dK1d** ()

  Constructor.

- virtual ~**HxImgFtorGenConv2dK1d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.108.1 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.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

8.108.2 Member Typedef Documentation

8.108.2.1 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. 761).

8.108.3 Constructor & Destructor Documentation

8.108.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv2dK1d () [inline]

Constructor.

```
26         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
27             HxImgFtorGenConvK1dKey(HxClassName<DstImgSigT>(),
28                 HxClassName<SrcImgSigT>(), HxClassName<KerImgSigT>(),
29                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
30                 HxClassName<KernelT>()))
31 {
32 #ifdef CD_TRACE
33     HxEnvironment::instance()->outputStream()
34         << "HxImgFtorGenConv2dK1d::HxImgFtorGenConv2dK1d()" << STD_ENDL;
35 #endif
36 }
```

8.108.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv2dK1d ()

`[virtual]`

Destructor.

```
43 {
44 #ifdef CD_TRACE
45     HxEnvironment::instance()->outputStream()
46         << "HxImgFtorGenConv2dK1d::~HxImgFtorGenConv2dK1d()" << STD_ENDL;
47 #endif
48 }
```

8.108.4 Member Function Documentation

8.108.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv2dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0)

`[protected, virtual]`

Do it.

Parameters:

dstPtr	Output image: IS = dstSize, IBS = 0
srcPtr	Input image: IS = srcSize, IBS = taglist(borderSize)
kerPtr	Input image, IS = kerSize, IBS = 0

Calls **HxFuncGenConv2dK1dDispatch** (p. 164) to dispatch the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
60 {
61     int dimension = HxGetTag<int>(tags, "dimension");
62     bool inplace = HxGetTag(tags, "inplace", false);
63
64     if (inplace && description)
65         description->setVariation("inplace");
66
67     HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
68     int borderWidth = borderSize.x();
```

```
69     int borderHeight = borderSize.y();
70     int kerWidth = kerSize.x();
71     if ((borderWidth == 0) && (dimension == 1))
72         borderWidth = kerWidth / 2;
73     if ((borderHeight == 0) && (dimension == 2))
74         borderHeight = kerWidth / 2;
75     srcPtr.incXYZ(borderWidth, borderHeight);
76
77     PixOpT pixOp(tags);
78     RedOpT redOp(tags);
79     KernelT kernel(kerPtr, kerSize, tags);
80
81     HxFuncGenConv2dK1dDispatch(dstPtr, srcPtr, kernel, dstSize,
82                                pixOp, redOp, dimension, inplace);
83 }
```

The documentation for this class was generated from the following files:

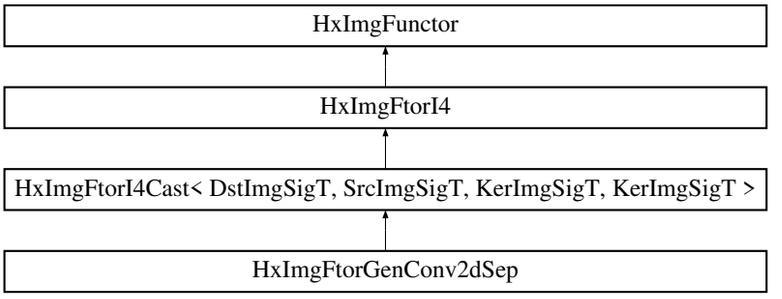
- **HxImgFtorGenConv2dK1d.h**
- HxImgFtorGenConv2dK1d.c

8.109 HxImgFtorGenConv2dSep Class Template Reference

Instantiation of generic algorithm for separable generalized convolution operations on 2d images.

`#include <HxImgFtorGenConv2dSep.h>`

Inheritance diagram for HxImgFtorGenConv2dSep::



Public Types

- typedef **HxImgFtorGenConv2dSepKey KeyType**
  The key type of this class.

Public Methods

- **HxImgFtorGenConv2dSep** ()
  Constructor.
- virtual ~**HxImgFtorGenConv2dSep** ()
  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** ker1Ptr, **Img4DataPtrType** ker2Ptr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** ker1Size, **HxSizes** ker2Size, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.109.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> class HxImgFtorGenConv2dSep< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >

Instantiation of generic algorithm for separable generalized convolution operations on 2d images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel images
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

8.109.2 Member Typedef Documentation

8.109.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> typedef HxImgFtorGenConv2dSepKey HxImgFtorGenConv2dSep::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI4Cast** (p. 773).

8.109.3 Constructor & Destructor Documentation

8.109.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2dSep< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv2dSep () [inline]

Constructor.

```
25         : HxImgFtorI4Cast<DstImgSigT, SrcImgSigT, KerImgSigT, KerImgSigT>(
26             HxImgFtorGenConv2dSepKey(
27                 HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
28                 HxClassName<KerImgSigT>(), HxClassName<KerImgSigT>(),
29                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
30                 HxClassName<KernelT>()))
31 {
32 #ifdef CD_TRACE
33     HxEnvironment::instance()->outputStream()
34         << "HxImgFtorGenConv2dSep::HxImgFtorGenConv2dSep()" << STD_ENDL;
```

```
35 #endif
36 }
```

8.109.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv2dSep< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv2dSep ()

[virtual]

Destructor.

```
43 {
44 #ifdef CD_TRACE
45     HxEnvironment::instance()->outputStream()
46         << "HxImgFtorGenConv2dSep::~HxImgFtorGenConv2dSep()" << STD_ENDL;
47 #endif
48 }
```

8.109.4 Member Function Documentation

8.109.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv2dSep< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *ker1Ptr*, Img4DataPtrType *ker2Ptr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *ker1Size*, HxSizes *ker2Size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0)

[protected, virtual]

Do it.

Assertions:

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = (ker1_NBW,ker2_NBW)

ker1Ptr Input image, IS = ker1Size, IBS = 0

ker2Ptr Input image, IS = ker2Size, IBS = 0

Calls **HxFuncGenConv2dSepDispatch** (p. 180) to dispatch the actual work.

Reimplemented from **HxImgFtorI4Cast** (p. 775).

```
62 {
63     int vType = HxGetTag(tags, "vType", 6);
64
65     if (description) {
66         HxString v("vType = ");
67         v += makeString(vType);
68         description->setVariation(v);
69     }
70
71     PixOpT pixOp(tags);
72     RedOpT redOp(tags);
73     KernelT kernel1(ker1Ptr, ker1Size, tags);
74     KernelT kernel2(ker2Ptr, ker2Size, tags);
75
76     HxFuncGenConv2dSepDispatch(dstPtr, srcPtr, kernel1, kernel2, dstSize,
77                                srcSize, pixOp, redOp, vType);
78 }
```

The documentation for this class was generated from the following files:

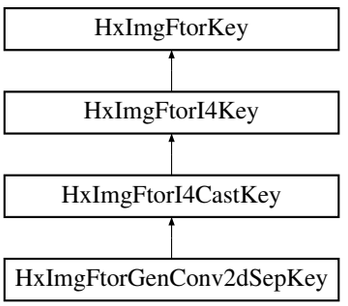
- **HxImgFtorGenConv2dSep.h**
- HxImgFtorGenConv2dSep.c

8.110 HxImgFtorGenConv2dSepKey Class Reference

Key for **HxImgFtorGenConv2dSep** (p. 717).

`#include <HxImgFtorGenConv2dSepKey.h>`

Inheritance diagram for HxImgFtorGenConv2dSepKey::



Public Methods

- **HxImgFtorGenConv2dSepKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** krnImg1Sig, **HxString** krnImg2Sig, **HxString** pixOpName, **HxString** redOpName, **HxString** kerName)

  Constructor.

8.110.1 Detailed Description

Key for **HxImgFtorGenConv2dSep** (p. 717).

8.110.2 Constructor & Destructor Documentation

8.110.2.1 HxImgFtorGenConv2dSepKey::HxImgFtorGenConv2dSepKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *krnImg1Sig*, HxString *krnImg2Sig*, HxString *pixOpName*, HxString *redOpName*, HxString *kerName*) [inline]

Constructor.

```
35     : HxImgFtorI4CastKey("HxImgFtorGenConv2dSep", dstImgSig, srcImgSig,
36                                                  krnImg1Sig, krnImg2Sig)
37 {
38     addArgument(pixOpName);
39     addArgument(redOpName);
40     addArgument(kerName);
41 }
```

The documentation for this class was generated from the following file:

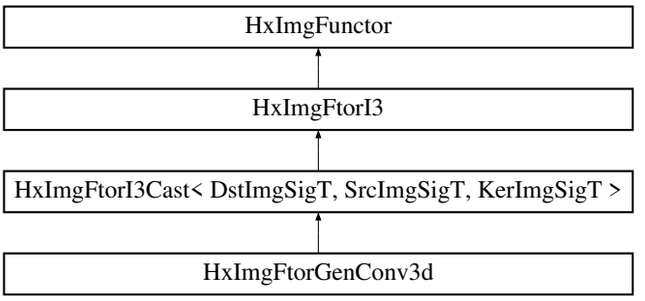
- **HxImgFtorGenConv2dSepKey.h**

8.111 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.

Public Methods

- **HxImgFtorGenConv3d** ()

  Constructor.

- virtual ∼**HxImgFtorGenConv3d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** ∗description=0)

  Do it.

8.111.1 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.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

8.111.2 Member Typedef Documentation

8.111.2.1 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. 761).

8.111.3 Constructor & Destructor Documentation

8.111.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv3d () [inline]

Constructor.

```
33         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
34             HxImgFtorGenConvKey(HxClassName<DstImgSigT>(),
35                 HxClassName<SrcImgSigT>(), HxClassName<KerImgSigT>(),
36                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
37                 HxClassName<KernelT>()))
38 {
39 #ifdef CD_TRACE
40     HxEnvironment::instance()->outputStream()
41         << "HxImgFtorGenConv3d::HxImgFtorGenConv3d()" << STD_ENDL;
42 #endif
43 }
```

8.111.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv3d () [virtual]

Destructor.

```
50 {
51 #ifdef CD_TRACE
52     HxEnvironment::instance()->outputStream()
53         << "HxImgFtorGenConv3d::~HxImgFtorGenConv3d()" << STD_ENDL;
54 #endif
55 }
```

8.111.4 Member Function Documentation

8.111.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv3d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = kerSize/2

kerPtr Input image, IS = kerSize, IBS = 0

Calls **HxFuncGenConv3dDispatch** (p. 181) to dispatch the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
65 {
66     bool rowpixfunc = HxGetTag(tags, "rowpixfunc", false);
67
68     if (rowpixfunc && description)
69         description->setVariation("rowpixfunc");
70
71     PixOpT pixOp(tags);
72     RedOpT redOp(tags);
73     KernelT kernel(kerPtr, kerSize, tags);
74
75     HxFuncGenConv3dDispatch(dstPtr, srcPtr, kernel, dstSize,
76                             pixOp, redOp, rowpixfunc);
77 }
```

The documentation for this class was generated from the following files:

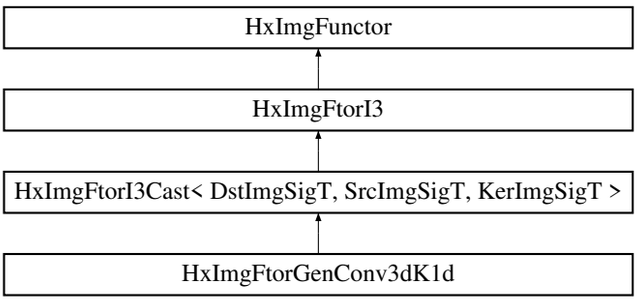
- **HxImgFtorGenConv3d.h**
- HxImgFtorGenConv3d.c

8.112 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.

Public Methods

- **HxImgFtorGenConv3dK1d** ()

  Constructor.

- virtual ~**HxImgFtorGenConv3dK1d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.112.1 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.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor
- KernelT is the type of the kernel functor

8.112.2 Member Typedef Documentation

8.112.2.1 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. 761).

8.112.3 Constructor & Destructor Documentation

8.112.3.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::HxImgFtorGenConv3dK1d () [inline]

Constructor.

```
33         : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, KerImgSigT>(
34             HxImgFtorGenConvK1dKey(
35                 HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
36                 HxClassName<KerImgSigT>(),
37                 HxClassName<PixOpT>(), HxClassName<RedOpT>(),
38                 HxClassName<KernelT>()))
39 {
40 #ifdef CD_TRACE
41     HxEnvironment::instance()->outputStream()
42         << "HxImgFtorGenConv3dK1d::HxImgFtorGenConv3dK1d()" << STD_ENDL;
43 #endif
44 }
```

8.112.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::~HxImgFtorGenConv3dK1d () [virtual]

Destructor.

```
51 {
52 #ifdef CD_TRACE
53     HxEnvironment::instance()->outputStream()
54         << "HxImgFtorGenConv3dK1d::~HxImgFtorGenConv3dK1d()" << STD_ENDL;
55 #endif
56 }
```

8.112.4 Member Function Documentation

8.112.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class PixOpT, class RedOpT, class KernelT> void HxImgFtorGenConv3dK1d< DstImgSigT, SrcImgSigT, KerImgSigT, PixOpT, RedOpT, KernelT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = taglist(borderSize)

kerPtr Input image, IS = kerSize, IBS = 0

Calls **HxFuncGenConv3dK1dDispatch** (p. 184) to dispatch the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
68 {
69     int dimension = HxGetTag<int>(tags, "dimension");
70
71     HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
72     int borderWidth = borderSize.x();
73     int borderHeight = borderSize.y();
74     int borderDepth = borderSize.z();
75     int kerWidth = kerSize.x();
76     if (dimension == 1) {
77         if (borderWidth == 0)
78             borderWidth = kerWidth / 2;
79         srcPtr.incXYZ(0, borderHeight, borderDepth);
80     }
81     if (dimension == 2) {
82         if (borderHeight == 0)
83             borderHeight = kerWidth / 2;
84         srcPtr.incXYZ(borderWidth, 0, borderDepth);
85     }
86     if (dimension == 3) {
87         if (borderDepth == 0)
88             borderDepth = kerWidth / 2;
89         srcPtr.incXYZ(borderWidth, borderHeight, 0);
90     }
91
92     PixOpT pixOp(tags);
93     RedOpT redOp(tags);
94     KernelT kernel(kerPtr, kerSize, tags);
95
96     HxFuncGenConv3dK1dDispatch(dstPtr, srcPtr, kernel, dstSize,
97                                pixOp, redOp, dimension);
98 }
```

The documentation for this class was generated from the following files:

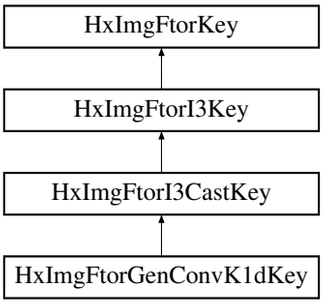
- **HxImgFtorGenConv3dK1d.h**
- HxImgFtorGenConv3dK1d.c

8.113 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.

8.113.1 Detailed Description

Key for HxImgFtorGenConvK1d.

8.113.2 Constructor & Destructor Documentation

8.113.2.1 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:

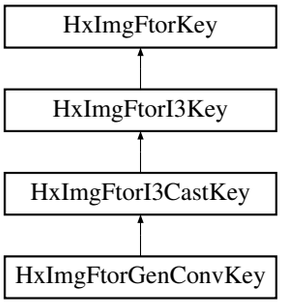
- **HxImgFtorGenConvK1dKey.h**

8.114 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.

8.114.1 Detailed Description

Key for HxImgFtorGenConv.

8.114.2 Constructor & Destructor Documentation

8.114.2.1 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:

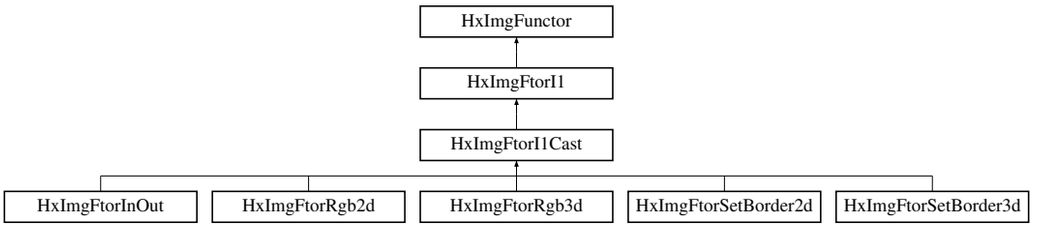
- **HxImgFtorGenConvKey.h**

8.115 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** (**HxImageData** *img, **HxTagList** &tags)=0
  callIt is implemented by **HxImgFtorI1Cast::callIt** (p. 734).

8.115.1 Detailed Description

Base class for image functors with one image parameter.

8.115.2 Member Typedef Documentation

8.115.2.1 typedef HxImgFtorI1Key HxImgFtorI1::KeyType

The key type of this class.

Reimplemented in **HxImgFtorI1Cast** (p. 732), **HxImgFtorInOut** (p. 793), **HxImgFtorRgb2d** (p. 836), **HxImgFtorRgb3d** (p. 838), **HxImgFtorSetBorder2d** (p. 850), **HxImgFtorSetBorder3d** (p. 853), **HxImgFtorInOut< ImgSigT, HxImageDataToHxMatrix< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign< typename ImgSigT::ArithType, typename**

ImgSigT::ArithType > > > (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutHistogram< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorRgb2d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb3d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), and **HxImgFtorRgb3d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838).

8.115.3 Constructor & Destructor Documentation

8.115.3.1 HxImgFtorI1::HxImgFtorI1 (const KeyType & *key*) [inline]

Constructor.

```
45                                          : HxImgFunctor(key)
46 {
47 }
```

8.115.3.2 HxImgFtorI1::~HxImgFtorI1 () [inline, virtual]

Destructor.

```
51 {
52 }
```

8.115.4 Member Function Documentation

8.115.4.1 virtual void HxImgFtorI1::callIt (HxImageData * *img*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI1Cast::callIt** (p. 734).

Reimplemented in **HxImgFtorI1Cast** (p. 734).

The documentation for this class was generated from the following file:

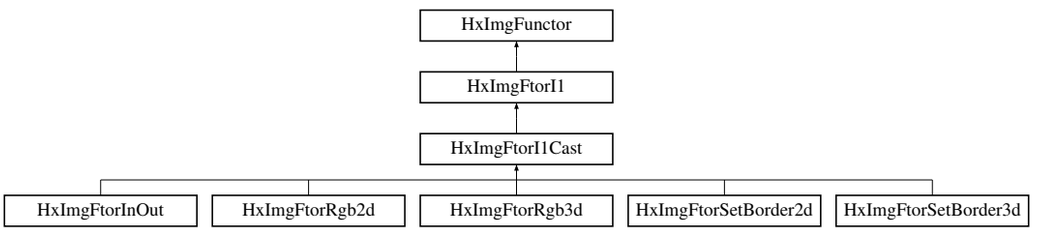
- **HxImgFtorI1.h**

8.116 HxImgFtorI1Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI1** (p. 728) 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** (**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:.

8.116.1 Detailed Description

template<class ImgSigT> class HxImgFtorI1Cast< ImgSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI1** (p. 728) to (statically typed) image data pointers.

Template parameters:

- ImgSigT is the signature type of the image

8.116.2 Member Typedef Documentation

8.116.2.1 template<class ImgSigT> typedef HxImgFtorI1CastKey HxImgFtorI1Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1** (p. 729).

Reimplemented in **HxImgFtorInOut** (p. 793), **HxImgFtorRgb2d** (p. 836), **HxImgFtorRgb3d** (p. 838), **HxImgFtorSetBorder2d** (p. 850), **HxImgFtorSetBorder3d** (p. 853), **HxImgFtorInOut< ImgSigT, HxImageDataToHxMatrix< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT,**

HxInOutHistogram< typename ImgSigT::ArithType > > (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorRgb2d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb2d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 836), **HxImgFtorRgb3d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), **HxImgFtorRgb3d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838), and **HxImgFtorRgb3d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 838).

8.116.2.2 template<class ImgSigT> typedef ImgSigT::DataPtrType HxImgFtorI1Cast::ImgDataPtrType

The data pointer type of the image.

8.116.3 Constructor & Destructor Documentation

8.116.3.1 template<class ImgSigT> HxImgFtorI1Cast< ImgSigT >::HxImgFtorI1Cast (const KeyType & *key*) [inline]

Constructor.

```
69                                                      : HxImgFtorI1(key)
70 {
71 }
```

8.116.3.2 template<class ImgSigT> HxImgFtorI1Cast< ImgSigT >::~HxImgFtorI1Cast () [virtual]

Destructor.

```
20 {
21 }
```

8.116.4 Member Function Documentation

8.116.4.1 template<class ImgSigT> void HxImgFtorI1Cast< ImgSigT >::callIt (HxImageData * *img*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI1** (p. 731).

```
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 }
```

8.116.4.2 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:.

- **HxImgFtorInOut::doIt** (p. 793)
- **HxImgFtorRgb2d::doIt** (p. 837)
- **HxImgFtorRgb3d::doIt** (p. 839)
- **HxImgFtorSetBorder2d::doIt** (p. 851)
- **HxImgFtorSetBorder3d::doIt** (p. 853)

Reimplemented in **HxImgFtorInOut** (p. 793), **HxImgFtorRgb2d** (p. 837), **HxImgFtorRgb3d** (p. 839), **HxImgFtorSetBorder2d** (p. 851), **HxImgFtorSetBorder3d** (p. 853), **HxImgFtorInOut< ImgSigT, HxImageDataToHxMatrix< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoSupAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType >**

> > (p. 793), **HxImgFtorInOut< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMulAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoAddAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMaxAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxInOutHistogram< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoMinAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > >** (p. 793), **HxImgFtorInOut< ImgSigT, HxReduceAdaptor< HxBpoInfAssign< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 793), **HxImgFtorRgb2d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb2d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 837), **HxImgFtorRgb3d< ImgSigT, HxRgbLab< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbLuv< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbStretch< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbOOO< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbLogMag< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbXYZ< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbBinary< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbLabel< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbHSI< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), **HxImgFtorRgb3d< ImgSigT, HxRgbDirect< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839), and **HxImgFtorRgb3d< ImgSigT, HxRgbCMY< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > >** (p. 839).

The documentation for this class was generated from the following files:

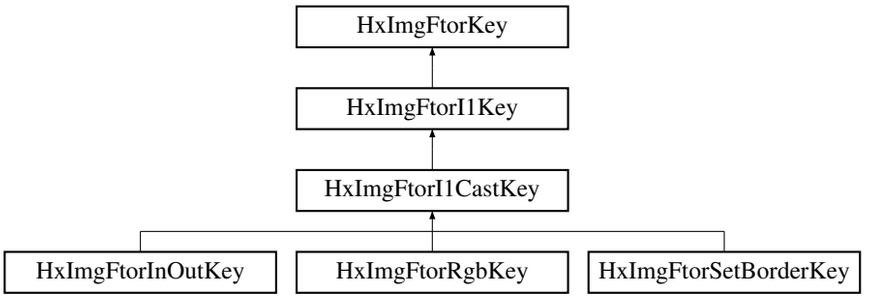
- **HxImgFtorI1Cast.h**
- HxImgFtorI1Cast.c

8.117 HxImgFtorI1CastKey Class Reference

Key for **HxImgFtorI1Cast** (p. 731).

`#include <HxImgFtorI1CastKey.h>`

Inheritance diagram for HxImgFtorI1CastKey::



Public Methods

- **HxImgFtorI1CastKey** (**HxString** className, **HxString** imgSig)
  Constructor.

8.117.1 Detailed Description

Key for **HxImgFtorI1Cast** (p. 731).

8.117.2 Constructor & Destructor Documentation

8.117.2.1 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:

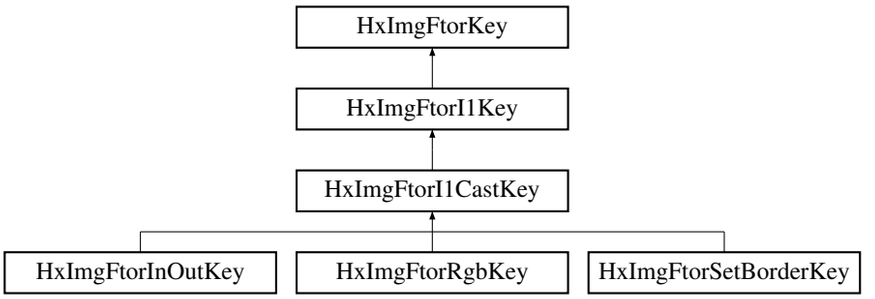
- **HxImgFtorI1CastKey.h**

8.118 HxImgFtorI1Key Class Reference

Key for **HxImgFtorI1** (p. 728).

`#include <HxImgFtorI1Key.h>`

Inheritance diagram for HxImgFtorI1Key::



Public Methods

- **HxImgFtorI1Key** (**HxString** className, **HxString** imgSig)

  Constructor.

8.118.1 Detailed Description

Key for **HxImgFtorI1** (p. 728).

8.118.2 Constructor & Destructor Documentation

8.118.2.1 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**

8.119 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** *img1, **HxImageData** *img2, **HxTagList** &tags)=0
  callIt is implemented by **HxImgFtorI2Cast::callIt** (p. 749).

8.119.1 Detailed Description

Base class for image functors with two image parameters.

8.119.2 Member Typedef Documentation

8.119.2.1 typedef HxImgFtorI2Key HxImgFtorI2::KeyType

The key type of this class.

Reimplemented in **HxImgFtorDiy** (p. 706), **HxImgFtorExportExtra** (p. 709), **HxImgFtorI2Cast** (p. 745), **HxImgFtorNgb2d** (p. 812), **HxImgFtorRecGenConv2d** (p. 829), **HxImgFtorRecGenConv2dK1d** (p. 831), **HxImgFtorSet** (p. 848), **HxImgFtorUpo** (p. 860), **HxImgFtorDiy< ImgSigT, ImgSigT, HxDiyTranspose< typename ImgSigT::DataPtrType, typename ImgSigT::DataPtrType > >** (p. 706), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMean< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraHistogram< typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskSum< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMedian< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraWeightMaskSum< typename ExtraImSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskStdev< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskCentralMoments< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMoments< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorI2Cast< DstSigT, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dFloat, SrcSigT >** (p. 745),

HxImgFtorI2Cast< HxImageSig2dByte, SrcSigT > (p. 745), **HxImgFtorI2Cast< ImgSigT, ImgSigT >** (p. 745), **HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig3dDouble, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig3dByte, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec3Float, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec2Float, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dDouble, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig3dInt, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec2Byte, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec3Byte, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dInt, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec3Double, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec3Short, SrcSigT >** (p. 745), **HxImgFtorI2Cast< DstSigT, ImgSigT >** (p. 745), **HxImgFtorI2Cast< ResSigT, ImgSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec2Double, SrcSigT >** (p. 745), **HxImgFtorI2Cast< ImgSigT, ExtraImSigT >** (p. 745), **HxImgFtorI2Cast< ImgSigT, KerSigT >** (p. 745), **HxImgFtorI2Cast< ImgSigT, ExtraImgSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig3dShort, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dShort, SrcSigT >** (p. 745), **HxImgFtorI2Cast< SrcSigT, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec2Int, SrcSigT >** (p. 745), **HxImgFtorI2Cast< ImgSigD, ImgSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec3Int, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dVec2Short, SrcSigT >** (p. 745), **HxImgFtorI2Cast< ImgSigT, KerImgSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig2dComplex, SrcSigT >** (p. 745), **HxImgFtorI2Cast< HxImageSig3dFloat, SrcSigT >** (p. 745), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbKuwahara< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d< typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbLWshed2d< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbDefuz< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbHilditch< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean< typename ResSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbBernsen< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d< typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d< typename DstSigT::ArithType, typename SrcSigT::ArithType > >** (p. 812), **HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 829), **HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 829), **HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 831), **HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 831), **HxImgFtorSet< HxImageSig2dFloat, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dByte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dDouble, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dByte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Float, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Float, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dDouble, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dInt, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Byte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Byte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dInt, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Double, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Short, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Double, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dShort, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dShort, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Int, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Int, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Short, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dComplex, SrcSigT >** (p. 848), **HxImgFtorSet< HxImage-**

Sig3dFloat, SrcSigT > (p. 848), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< HxImageSig3dInt, SrcSigT, HxUpoCopy< typename HxImageSig3dInt::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, RGB2Intensity< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoSum< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSinh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< HxImageSig3dByte, SrcSigT, HxUpoCopy< typename HxImageSig3dByte::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoConjugate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAsin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoColSpace< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNormInf< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig3dFloat, SrcSigT, HxUpoCopy< typename HxImageSig3dFloat::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dFloat, SrcSigT, HxUpoCopy< typename HxImageSig2dFloat::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoComplement< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dComplex, SrcSigT, HxUpoCopy<**

typename HxImageSig2dComplex::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< HxImageSig2dVec3Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Short::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Double::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMax< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< HxImageSig2dVec3Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Byte::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoNegate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCeil< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRound< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAtan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCosh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Float::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Short::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Byte::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< SrcSigT, SrcSigT, HxUpoBinMap< typename SrcSigT::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoProduct< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog10< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAcos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dInt, SrcSigT, HxUpoCopy< typename HxImageSig2dInt::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm1< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLUT< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoAtan2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTanh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMin< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigD, ImgSigT, HxUpoReciprocal< typename ImgSigD::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpo-

Bind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoExp< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec2Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Int::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSqrt< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoArg< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dByte, SrcSigT, HxUpoCopy< typename HxImageSig2dByte::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRAVBO< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoFloor< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec3Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Double::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2Sqr< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig3dDouble, SrcSigT, HxUpoCopy< typename HxImageSig3dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig3dShort, SrcSigT, HxUpoCopy< typename HxImageSig3dShort::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec3Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Float::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec3Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Int::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dDouble, SrcSigT, HxUpoCopy< typename HxImageSig2dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dShort, SrcSigT, HxUpoCopy< typename HxImageSig2dShort::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAbs< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSetPartImage< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), and **HxImgFtorUpo< ImgSigT, ImgSigT, AffinePix< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860).

8.119.3 Constructor & Destructor Documentation

8.119.3.1 HxImgFtorI2::HxImgFtorI2 (const KeyType & *key*) [inline]

Constructor.

```
48                                          : HxImgFunctor(key)
49 {
50 }
```

8.119.3.2 HxImgFtorI2::~HxImgFtorI2 () [inline, virtual]

Destructor.

```
54 {
55 }
```

8.119.4 Member Function Documentation

8.119.4.1 virtual void HxImgFtorI2::callIt (HxImageData * *img1*, HxImageData * *img2*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI2Cast::callIt** (p. 749).

Reimplemented in **HxImgFtorI2Cast** (p. 749), **HxImgFtorI2Cast< DstSigT, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dFloat, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dByte, SrcSigT >** (p. 749), **HxImgFtorI2Cast< ImgSigT, ImgSigT >** (p. 749), **HxImgFtorI2Cast< DstImgSigT, SrcImgSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig3dDouble, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig3dByte, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec3Float, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec2Float, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dDouble, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig3dInt, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec2Byte, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec3Byte, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dInt, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec3Double, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec3Short, SrcSigT >** (p. 749), **HxImgFtorI2Cast< DstSigT, ImgSigT >** (p. 749), **HxImgFtorI2Cast< ResSigT, ImgSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec2Double, SrcSigT >** (p. 749), **HxImgFtorI2Cast< ImgSigT, ExtraImSigT >** (p. 749), **HxImgFtorI2Cast< ImgSigT, KerSigT >** (p. 749), **HxImgFtorI2Cast< ImgSigT, ExtraImgSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig3dShort, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dShort, SrcSigT >** (p. 749), **HxImgFtorI2Cast< SrcSigT, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec2Int, SrcSigT >** (p. 749), **HxImgFtorI2Cast< ImgSigD, ImgSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec3Int, SrcSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dVec2Short, SrcSigT >** (p. 749), **HxImgFtorI2Cast< ImgSigT, KerImgSigT >** (p. 749), **HxImgFtorI2Cast< HxImageSig2dComplex, SrcSigT >** (p. 749), and **HxImgFtorI2Cast< HxImageSig3dFloat, SrcSigT >** (p. 749).

The documentation for this class was generated from the following file:

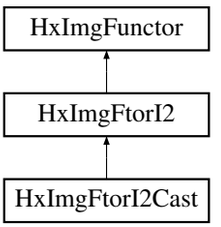
- **HxImgFtorI2.h**

8.120 HxImgFtorI2Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI2** (p. 737) 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 Img1SigT::DataPtrType **Img1DataPtrType**
  The data pointer type of the first image.
- typedef Img2SigT::DataPtrType **Img2DataPtrType**
  The data pointer type of the second image.

Public Methods

- **HxImgFtorI2Cast** (const **KeyType** &)
  Constructor.
- virtual ~**HxImgFtorI2Cast** ()
  Destructor.
- virtual void **callIt** (**HxImageData** *img1, **HxImageData** *img2, **HxTagList** &tags)
  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** img1Ptr, **Img2DataPtrType** img2Ptr, **HxSizes** img1Size, **HxSizes** img2Size, **HxTagList** &tags, **HxImgFtorDescription** *description=0)=0
  doIt is implemented by derived image functors:.

8.120.1 Detailed Description

template<class Img1SigT, class Img2SigT> class HxImgFtorI2Cast< Img1SigT, Img2SigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI2** (p. 737) to (statically typed) image data pointers.

Template parameters:

- Img1SigT is the signature type of the fist image
- Img2SigT is the signature type of the second image

8.120.2 Member Typedef Documentation

8.120.2.1 template<class Img1SigT, class Img2SigT> typedef HxImgFtorI2CastKey HxImgFtorI2Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2** (p. 738).

Reimplemented in **HxImgFtorDiy** (p. 706), **HxImgFtorExportExtra** (p. 709), **HxImgFtorNgb2d** (p. 812), **HxImgFtorRecGenConv2d** (p. 829), **HxImgFtorRecGenConv2dK1d** (p. 831), **HxImgFtorSet** (p. 848), **HxImgFtorUpo** (p. 860), **HxImgFtorDiy< ImgSigT, ImgSigT, HxDiyTranspose< typename ImgSigT::DataPtrType, typename ImgSigT::DataPtrType > >** (p. 706), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMean< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraHistogram< typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskSum< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMedian< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraWeightMaskSum< typename ExtraImSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskStdev< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskCentralMoments< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMoments< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 709), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbKuwahara< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d< typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbLWshed2d< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbDefuz< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbHilditch< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean< typename ResSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbBernsen< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d< typename ImgSigT::ArithType > >** (p. 812), **HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d< typename DstSigT::ArithType, typename SrcSigT::ArithType > >** (p. 812), **HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 829), **HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 829), **HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 831), **HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 831), **HxImgFtorSet< HxImageSig2dFloat, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dByte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dDouble, SrcSig-**

T > (p. 848), HxImgFtorSet< HxImageSig3dByte, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec3Float, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec2Float, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dDouble, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig3dInt, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec2Byte, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec3Byte, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dInt, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec3Double, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec3Short, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec2Double, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig3dShort, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dShort, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec2Int, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec3Int, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dVec2Short, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig2dComplex, SrcSigT > (p. 848), HxImgFtorSet< HxImageSig3dFloat, SrcSigT > (p. 848), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< HxImageSig3dInt, SrcSigT, HxUpoCopy< typename HxImageSig3dInt::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, RGB2Intensity< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoSum< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSinh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< HxImageSig3dByte, SrcSigT, HxUpoCopy< typename HxImageSig3dByte::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoConjugate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAsin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoColSpace< typename DstSigT::ArithTypeDouble, typename Img-

SigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNormInf< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig3dFloat, SrcSigT, HxUpoCopy< typename HxImageSig3dFloat::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dFloat, SrcSigT, HxUpoCopy< typename HxImageSig2dFloat::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoComplement< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dComplex, SrcSigT, HxUpoCopy< typename HxImageSig2dComplex::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< HxImageSig2dVec3Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Short::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Double::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMax< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo< HxImageSig2dVec3Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Byte::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoNegate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCeil< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRound< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAtan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCosh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Float::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Short::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dVec2Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Byte::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< SrcSigT, SrcSigT, HxUpoBinMap< typename SrcSigT::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoProduct< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog10< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAcos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< HxImageSig2dInt, SrcSigT, HxUpoCopy< typename HxImageSig2dInt::ArithType, typename SrcSigT::ArithType > > (p. 860), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm1< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 860), HxImgFtorUpo<

ImgSigT, ImgSigT, HxUpoLUT< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoAtan2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTanh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMin< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigD, ImgSigT, HxUpoReciprocal< typename ImgSigD::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoExp< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec2Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Int::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSqrt< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoArg< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dByte, SrcSigT, HxUpoCopy< typename HxImageSig2dByte::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRAVBO< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoFloor< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec3Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Double::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 860), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2Sqr< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig3dDouble, SrcSigT, HxUpoCopy< typename HxImageSig3dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig3dShort, SrcSigT, HxUpoCopy< typename HxImageSig3dShort::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec3Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Float::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dVec3Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Int::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dDouble, SrcSigT, HxUpoCopy< typename HxImageSig2dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< HxImageSig2dShort, SrcSigT, HxUpoCopy< typename HxImageSig2dShort::ArithType, typename SrcSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAbs< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSetPartImage< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860), and **HxImgFtorUpo< ImgSigT, ImgSigT, AffinePix< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 860).

8.120.2.2 template<class Img1SigT, class Img2SigT> typedef Img1SigT::DataPtrType HxImgFtorI2Cast::Img1DataPtrType

The data pointer type of the first image.

8.120.2.3 template<class Img1SigT, class Img2SigT> typedef Img2SigT::DataPtrType HxImgFtorI2Cast::Img2DataPtrType

The data pointer type of the second image.

8.120.3 Constructor & Destructor Documentation

8.120.3.1 template<class Img1SigT, class Img2SigT> HxImgFtorI2Cast< Img1SigT, Img2SigT >::HxImgFtorI2Cast (const KeyType & *key*) [inline]

Constructor.

```
80             : HxImgFtorI2(key)
81 {
82 }
```

8.120.3.2 template<class Img1SigT, class Img2SigT> HxImgFtorI2Cast< Img1SigT, Img2SigT >::~HxImgFtorI2Cast () [virtual]

Destructor.

```
26 {
27 #ifdef CD_TRACE
28     HxEnvironment::instance()->outputStream()
29         << "HxImgFtorI2Cast<>::~HxImgFtorI2Cast()" << STD_ENDL;
30     HxEnvironment::instance()->flush();
31 #endif
32 }
```

8.120.4 Member Function Documentation

8.120.4.1 template<class Img1SigT, class Img2SigT> void HxImgFtorI2Cast< Img1SigT, Img2SigT >::callIt (HxImageData * *img1*, HxImageData * *img2*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI2** (p. 743).

```
38 {
39     TYPENAME Img1SigT::DataPtrType img1Ptr
40         = HxMakeDataPtr<typename Img1SigT::DataPtrType>(img1);
41     TYPENAME Img2SigT::DataPtrType img2Ptr
42         = HxMakeDataPtr<typename Img2SigT::DataPtrType>(img2);
43     HxImgFtorDescription* description = getDescription();
44     if (description)
45     {
46         description->setTags(tags);
```

```
47          description->addArgument(img1->signature(), img1->sizes());
48          description->addArgument(img2->signature(), img2->sizes());
49          description->startTime();
50      }
51
52      doIt(
53          img1Ptr, img2Ptr, img1->sizes(), img2->sizes(),
54          tags, description);
55
56      if (description)
57          description->stopTime();
58  }
```

8.120.4.2 template<class Img1SigT, class Img2SigT> virtual void HxImgFtorI2Cast< Img1SigT, Img2SigT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors:.

- **HxImgFtorExportExtra::doIt** (p. 710)
- **HxImgFtorNgb2d::doIt** (p. 813)
- HxImgFtorRecNgb2d::doIt
- **HxImgFtorSet::doIt** (p. 848)
- HxImgFtorTranspose2d::doIt
- **HxImgFtorUpo::doIt** (p. 861)

Reimplemented in **HxImgFtorDiy** (p. 707), **HxImgFtorExportExtra** (p. 710), **HxImgFtorNgb2d** (p. 813), **HxImgFtorRecGenConv2d** (p. 830), **HxImgFtorRecGenConv2dK1d** (p. 832), **HxImgFtorSet** (p. 848), **HxImgFtorUpo** (p. 861), **HxImgFtorDiy< ImgSigT, ImgSigT, HxDiyTranspose< typename ImgSigT::DataPtrType, typename ImgSigT::DataPtrType > >** (p. 707), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMean< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraHistogram< typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskSum< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMedian< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraWeightMaskSum< typename ExtraImSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskStdev< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskCentralMoments< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMoments< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > >** (p. 710), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbKuwahara< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d< typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbLWshed2d< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbDefuz< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbHilditch< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean< typename**

ResSigT::ArithType, typename ImgSigT::ArithType > > (p. 813), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbBernsen< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d< typename ImgSigT::ArithType > >** (p. 813), **HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d< typename DstSigT::ArithType, typename SrcSigT::ArithType > >** (p. 813), **HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 830), **HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 830), **HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoMul< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoAddAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 832), **HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType > >** (p. 832), **HxImgFtorSet< HxImageSig2dFloat, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dByte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dDouble, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dByte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Float, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Float, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dDouble, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dInt, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Byte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Byte, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dInt, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Double, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Short, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Double, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dShort, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dShort, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Int, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec3Int, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dVec2Short, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig2dComplex, SrcSigT >** (p. 848), **HxImgFtorSet< HxImageSig3dFloat, SrcSigT >** (p. 848), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< HxImageSig3dInt, SrcSigT, HxUpoCopy< typename HxImageSig3dInt::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< DstSigT, ImgSigT, RGB2Intensity< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 861), **HxImgFtorUpo< DstSigT, ImgSigT, HxUpoSum< typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 861), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSinh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 861), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< HxImageSig3dByte, SrcSigT, HxUpoCopy< typename HxImageSig3dByte::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > >** (p. 861), **HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, type-**

name ImgSigT::ArithType, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoConjugate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAsin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoColSpace< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNormInf< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig3dFloat, SrcSigT, HxUpoCopy< typename HxImageSig3dFloat::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dFloat, SrcSigT, HxUpoCopy< typename HxImageSig2dFloat::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoComplement< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dComplex, SrcSigT, HxUpoCopy< typename HxImageSig2dComplex::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTriStateThreshold< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< HxImageSig2dVec3Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Short::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dVec2Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Double::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMax< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< HxImageSig2dVec3Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Byte::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoNegate< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoCeil< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRound< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAtan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSig-

T, ImgSigT, HxUpoCosh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dVec2Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Float::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dVec2Short, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Short::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dVec2Byte, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Byte::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< SrcSigT, SrcSigT, HxUpoBinMap< typename SrcSigT::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoProduct< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLog10< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAcos< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dInt, SrcSigT, HxUpoCopy< typename HxImageSig2dInt::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm1< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoLUT< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoAtan2< typename DstSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTanh< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoMin< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigD, ImgSigT, HxUpoReciprocal< typename ImgSigD::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSin< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoExp< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dVec2Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec2Int::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSqrt< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoArg< typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dByte, SrcSigT, HxUpoCopy< typename HxImageSig2dByte::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoTan< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxBpoBind2Val< typename DstSigT::ArithType, typename ImgSigT::ArithType, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoRAVBO< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoFloor< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 861), HxImgFtorUpo< HxImageSig2dVec3Double, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Double::ArithType, typename SrcSigT::ArithType > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< ImgSigT, ImgSigT, HxBpoBind2Val< typename ImgSigT::ArithType, typename ImgSigT::ArithType, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > > (p. 861), HxImgFtorUpo< DstSigT, ImgSigT, HxUpoNorm2Sqr< typename DstSig-

T::ArithType, typename ImgSigT::ArithType > > (p. 861), **HxImgFtorUpo< HxImageSig3dDouble, SrcSigT, HxUpoCopy< typename HxImageSig3dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< HxImageSig3dShort, SrcSigT, HxUpoCopy< typename HxImageSig3dShort::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< HxImageSig2dVec3Float, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Float::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< HxImageSig2dVec3Int, SrcSigT, HxUpoCopy< typename HxImageSig2dVec3Int::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< HxImageSig2dDouble, SrcSigT, HxUpoCopy< typename HxImageSig2dDouble::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< HxImageSig2dShort, SrcSigT, HxUpoCopy< typename HxImageSig2dShort::ArithType, typename SrcSigT::ArithType > >** (p. 861), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoAbs< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 861), **HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSetPartImage< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 861), and **HxImgFtorUpo< ImgSigT, ImgSigT, AffinePix< typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 861).

The documentation for this class was generated from the following files:

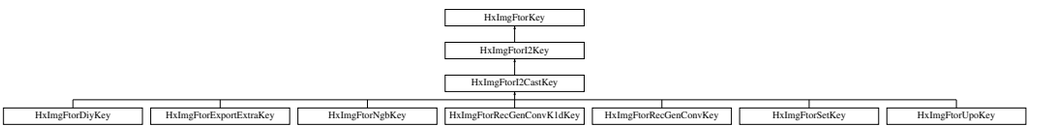
- **HxImgFtorI2Cast.h**
- HxImgFtorI2Cast.c

8.121 HxImgFtorI2CastKey Class Reference

Key for **HxImgFtorI2Cast** (p. 743).

`#include <HxImgFtorI2CastKey.h>`

Inheritance diagram for HxImgFtorI2CastKey::



Public Methods

- **HxImgFtorI2CastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

8.121.1 Detailed Description

Key for **HxImgFtorI2Cast** (p. 743).

8.121.2 Constructor & Destructor Documentation

8.121.2.1 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:

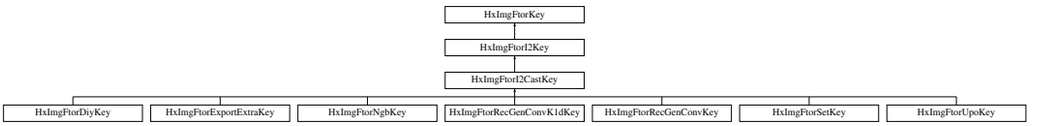
- **HxImgFtorI2CastKey.h**

8.122 HxImgFtorI2Key Class Reference

Key for **HxImgFtorI2** (p. 737).

`#include <HxImgFtorI2Key.h>`

Inheritance diagram for HxImgFtorI2Key::



Public Methods

- **HxImgFtorI2Key** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

8.122.1 Detailed Description

Key for **HxImgFtorI2** (p. 737).

8.122.2 Constructor & Destructor Documentation

8.122.2.1 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**

8.123 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** *img1, **HxImageData** *img2, **HxImageData** *img3, **HxTagList** &tags)=0

  callIt is implemented by **HxImgFtorI3Cast::callIt** (p. 764).

8.123.1 Detailed Description

Base class for image functors with three image parameters.

8.123.2 Member Typedef Documentation

8.123.2.1 typedef HxImgFtorI3Key HxImgFtorI3::KeyType

The key type of this class.

Reimplemented in **HxImgFtorBpo** (p. 701), **HxImgFtorGenConv2d** (p. 713), **HxImgFtorGenConv2dK1d** (p. 715), **HxImgFtorGenConv3d** (p. 722), **HxImgFtorGenConv3dK1d** (p. 725), **HxImgFtorI3Cast** (p. 761), **HxImgFtorKernelNgb2d** (p. 796), **HxImgFtorNgb2dExtra** (p. 814), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoXor<**

typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxMagnitude< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, LblSigT, HxBpoHighlightRegion< typename ImgSigT::ArithType, typename ImgSigT::ArithTypeDouble, typename LblSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAddSat< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSubSat< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), 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. 713), 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. 713), 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. 713), 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. 713), 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. 713), 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. 715), 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. 715), 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. 715), 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. 715), 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. 715), 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. 722), 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. 725), HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT > (p. 761), HxImgFtorI3Cast< ImgSigT, ImgSigT, LblSigT > (p. 761), HxImgFtorI3Cast< DstSigT, SrcSigT, KerSigT > (p. 761), HxImgFtorI3Cast< ImgSigT, ExtraSigT, AnotherSigT > (p. 761), HxImgFtorI3Cast< DstSigT, ImgSigT, ImgSigT > (p. 761), HxImgFtorI3Cast< ImgSigT, ImgSigT, ImgSigT > (p. 761), HxImgFtorI3Cast< DstImgSigT, SrcImgSigT, KerImgSigT > (p. 761), HxImgFtorI3Cast< ImgSigT, ExtraSigT, MaskImgSigT > (p. 761), HxImgFtorI3Cast< InOutT, InOutT, InOutT > (p. 761), HxImgFtorI3Cast< ImgSigT, KerSigT, KerSigT > (p. 761), HxImgFtorI3Cast< DstSigT, DstSigT, ImgSigT > (p. 761), HxImgFtorI3Cast< DstImgSigT, SrcImgSigT, ExtraImgSigT > (p. 761), HxImgFtorKernelNgb2d< DstSigT, SrcSigT, KerSigT, HxKerNgbNormCorrelation< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > > (p. 796), and HxImgFtorNgb2dExtra< InOutT, InOutT, InOutT, HxNgbLocalMode< typename InOutT::ArithType, typename InOutT::ArithType > > (p. 814).

8.123.3 Constructor & Destructor Documentation

8.123.3.1 HxImgFtorI3::HxImgFtorI3 (const KeyType & *key*) [inline]

Constructor.

```
48                                              : HxImgFunctor(key)
49 {
50 }
```

8.123.3.2 HxImgFtorI3::~HxImgFtorI3 () [inline, virtual]

Destructor.

```
54 {
55 }
```

8.123.4 Member Function Documentation

8.123.4.1 virtual void HxImgFtorI3::callIt (HxImageData * *img1*, HxImageData * *img2*, HxImageData * *img3*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI3Cast::callIt** (p. 764).

Reimplemented in **HxImgFtorI3Cast** (p. 764), **HxImgFtorI3Cast< DstImgSigT, Src1ImgSigT, Src2ImgSigT >** (p. 764), **HxImgFtorI3Cast< ImgSigT, ImgSigT, LblSigT >** (p. 764), **HxImgFtorI3Cast< DstSigT, SrcSigT, KerSigT >** (p. 764), **HxImgFtorI3Cast< ImgSigT, ExtraSigT, AnotherSigT >** (p. 764), **HxImgFtorI3Cast< DstSigT, ImgSigT, ImgSigT >** (p. 764), **HxImgFtorI3Cast< ImgSigT, ImgSigT, ImgSigT >** (p. 764), **HxImgFtorI3Cast< DstImgSigT, SrcImgSigT, KerImgSigT >** (p. 764), **HxImgFtorI3Cast< ImgSigT, ExtraSigT, MaskImgSigT >** (p. 764), **HxImgFtorI3Cast< InOutT, InOutT, InOutT >** (p. 764), **HxImgFtorI3Cast< ImgSigT, KerSigT, KerSigT >** (p. 764), **HxImgFtorI3Cast< DstSigT, DstSigT, ImgSigT >** (p. 764), and **HxImgFtorI3Cast< DstImgSigT, SrcImgSigT, ExtraImgSigT >** (p. 764).

The documentation for this class was generated from the following file:

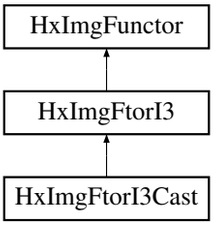
- **HxImgFtorI3.h**

8.124 HxImgFtorI3Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI3** (p. 756) 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 Img1SigT::DataPtrType **Img1DataPtrType**

  The data pointer type of the first image.

- typedef Img2SigT::DataPtrType **Img2DataPtrType**

  The data pointer type of the second image.

- typedef Img3SigT::DataPtrType **Img3DataPtrType**

  The data pointer type of the third image.

Public Methods

- **HxImgFtorI3Cast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI3Cast** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *img1, **HxImageData** *img2, **HxImageData** *img3, **HxTagList** &tags)

  Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** img1Ptr, **Img2DataPtrType** img2Ptr, **Img3DataPtrType** img3Ptr, **HxSizes** img1Size, **HxSizes** img2Size, **HxSizes** img3Size, **HxTagList** &tags, **HxImgFtorDescription** *=0)=0

  doIt is implemented by derived image functors:.

8.124.1 Detailed Description

template<class Img1SigT, class Img2SigT, class Img3SigT> class HxImgFtorI3Cast< Img1SigT, Img2SigT, Img3SigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI3** (p. 756) to (statically typed) image data pointers.

Template parameters:

- Img1SigT is the signature type of the first image
- Img2SigT is the signature type of the second image
- Img3SigT is the signature type of the third image

8.124.2 Member Typedef Documentation

8.124.2.1 template<class Img1SigT, class Img2SigT, class Img3SigT> typedef HxImgFtorI3CastKey HxImgFtorI3Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3** (p. 756).

Reimplemented in **HxImgFtorBpo** (p. 701), **HxImgFtorGenConv2d** (p. 713), **HxImgFtorGenConv2dK1d** (p. 715), **HxImgFtorGenConv3d** (p. 722), **HxImgFtorGenConv3dK1d** (p. 725), **HxImgFtorKernelNgb2d** (p. 796), **HxImgFtorNgb2dExtra** (p. 814), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxMagnitude< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 701), **HxImgFtorBpo< ImgSigT, Img-**

SigT, ImgSigT, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, LblSigT, HxBpoHighlightRegion< typename ImgSigT::ArithType, typename ImgSigT::ArithTypeDouble, typename LblSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAddSat< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSubSat< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 701), 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. 713), 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. 713), 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. 713), 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. 713), 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. 713), 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. 715), 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. 715), 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. 715), HxImgFtorGenConv2dK1d< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::Arith-

Type, typename KerSigT::ArithType, typename KerSigT::ArithType >, HxBpoMinAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > (p. 715), **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. 715), **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. 722), **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. 725), **HxImgFtorKernelNgb2d< DstSigT, SrcSigT, KerSigT, HxKerNgbNormCorrelation< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > >** (p. 796), and **HxImgFtorNgb2dExtra< InOutT, InOutT, InOutT, HxNgbLocalMode< typename InOutT::ArithType, typename InOutT::ArithType > >** (p. 814).

8.124.2.2 template<class Img1SigT, class Img2SigT, class Img3SigT> typedef Img1SigT::DataPtrType HxImgFtorI3Cast::Img1DataPtrType

The data pointer type of the first image.

8.124.2.3 template<class Img1SigT, class Img2SigT, class Img3SigT> typedef Img2SigT::DataPtrType HxImgFtorI3Cast::Img2DataPtrType

The data pointer type of the second image.

8.124.2.4 template<class Img1SigT, class Img2SigT, class Img3SigT> typedef Img3SigT::DataPtrType HxImgFtorI3Cast::Img3DataPtrType

The data pointer type of the third image.

8.124.3 Constructor & Destructor Documentation

8.124.3.1 template<class Img1SigT, class Img2SigT, class Img3SigT> HxImgFtorI3Cast< Img1SigT, Img2SigT, Img3SigT >::HxImgFtorI3Cast (const KeyType & *key*) [inline]

Constructor.

```
85                                      : HxImgFtorI3(key)
86 {
87 }
```

8.124.3.2 template<class Img1SigT, class Img2SigT, class Img3SigT> HxImgFtorI3Cast< Img1SigT, Img2SigT, Img3SigT >::~HxImgFtorI3Cast () [virtual]

Destructor.

```
25 {
26 #ifdef CD_TRACE
27     HxEnvironment::instance()->outputStream()
28         << "~HxImgFtorI3Cast()" << STD_ENDL;
29     HxEnvironment::instance()->flush();
30 #endif
31 }
```

8.124.4 Member Function Documentation

8.124.4.1 template<class Img1SigT, class Img2SigT, class Img3SigT> void HxImgFtorI3Cast< Img1SigT, Img2SigT, Img3SigT >::callIt (HxImageData * *img1*, HxImageData * *img2*, HxImageData * *img3*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI3** (p. 759).

```
38 {
39     TYPENAME Img1SigT::DataPtrType img1Ptr
40         = HxMakeDataPtr<typename Img1SigT::DataPtrType>(img1);
41     TYPENAME Img2SigT::DataPtrType img2Ptr
42         = HxMakeDataPtr<typename Img2SigT::DataPtrType>(img2);
43     TYPENAME Img3SigT::DataPtrType img3Ptr
44         = HxMakeDataPtr<typename Img3SigT::DataPtrType>(img3);
45
46     HxImgFtorDescription* description = getDescription();
47     if (description)
48     {
49         description->setTags(tags);
50         description->addArgument(img1->signature(), img1->sizes());
51         description->addArgument(img2->signature(), img2->sizes());
52         description->addArgument(img3->signature(), img3->sizes());
53         description->startTime();
54     }
55
56     doIt(img1Ptr, img2Ptr, img3Ptr,
57         img1->sizes(), img2->sizes(), img3->sizes(), tags, description);
58
59     if (description)
60         description->stopTime();
61 }
```

8.124.4.2 template<class Img1SigT, class Img2SigT, class Img3SigT> virtual void HxImgFtorI3Cast< Img1SigT, Img2SigT, Img3SigT >::doIt (Img1DataPtrType *imgPtr*, Img2DataPtrType *extraPtr*, Img3DataPtrType *anotherPtr*, HxSizes *imgSize*, HxSizes *extraSize*, HxSizes *anotherSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors:.

- **HxImgFtorBpo::doIt** (p. 703)
- **HxImgFtorGenConv2d::doIt** (p. 713)
- **HxImgFtorGenConv2dK1d::doIt** (p. 716)
- **HxImgFtorGenConv3d::doIt** (p. 723)
- **HxImgFtorGenConv3dK1d::doIt** (p. 725)

- **HxImgFtorKernelNgb2d::doIt** (p. 797)

Reimplemented in **HxImgFtorBpo** (p. 703), **HxImgFtorGenConv2d** (p. 713), **HxImgFtorGenConv2dK1d** (p. 716), **HxImgFtorGenConv3d** (p. 723), **HxImgFtorGenConv3dK1d** (p. 725), **HxImgFtorKernelNgb2d** (p. 797), **HxImgFtorNgb2dExtra** (p. 815), **HxImgFtorQueueBased** (p. 824), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoXor< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMin< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSqrDst< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSub< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoInf< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoDot< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoOr< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxMagnitude< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAnd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSup< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoNotEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoCross< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoPow< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoRightShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoLeftShift< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoDiv< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMax< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoGreaterThan< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMul< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, LblSigT, HxBpoHighlightRegion< typename ImgSigT::ArithType, typename ImgSigT::ArithTypeDouble, typename LblSigT::ArithType > >** (p. 703), **HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAddSat< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoMod< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > >** (p. 703), **HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoSubSat< typename ImgSigT::ArithType, typename ImgSigT::ArithType, type-**

name ImgSigT::ArithType > > (p. 703), HxImgFtorBpo< ImgSigT, ImgSigT, ImgSigT, HxBpoAdd< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 703), HxImgFtorBpo< DstSigT, ImgSigT, ImgSigT, HxBpoLessEqual< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > (p. 703), 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. 713), 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. 713), 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. 713), 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. 713), 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. 713), 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. 716), 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. 716), 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. 716), 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. 716), 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. 716), 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. 723), 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. 725), HxImgFtorKernelNgb2d< DstSigT, SrcSigT, KerSigT, HxKerNgbNormCorrelation< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > > (p. 797), HxImgFtorNgb2dExtra< InOutT, InOutT, InOutT, HxNgbLocalMode< typename InOutT::ArithType, typename InOutT::ArithType > > (p. 815), HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, QWatershedMarkers2< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > > (p. 824), HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, QWatershedMarkers< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >

(p. 824), **HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, QLabelGrassFire< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >** (p. 824), **HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, ImageToSegmentation< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >** (p. 824), **HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, GetBlobFeatures< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >** (p. 824), **HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, QWaterShedLV< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >** (p. 824), **HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, QThinning< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >** (p. 824), and **HxImgFtorQueueBased< ImgSigT, ExtraSigT, MaskImgSigT, QLabelFrans< typename ImgSigT::ArithType, typename ExtraSigT::ArithType, typename MaskImgSigT::ArithType > >** (p. 824).

The documentation for this class was generated from the following files:

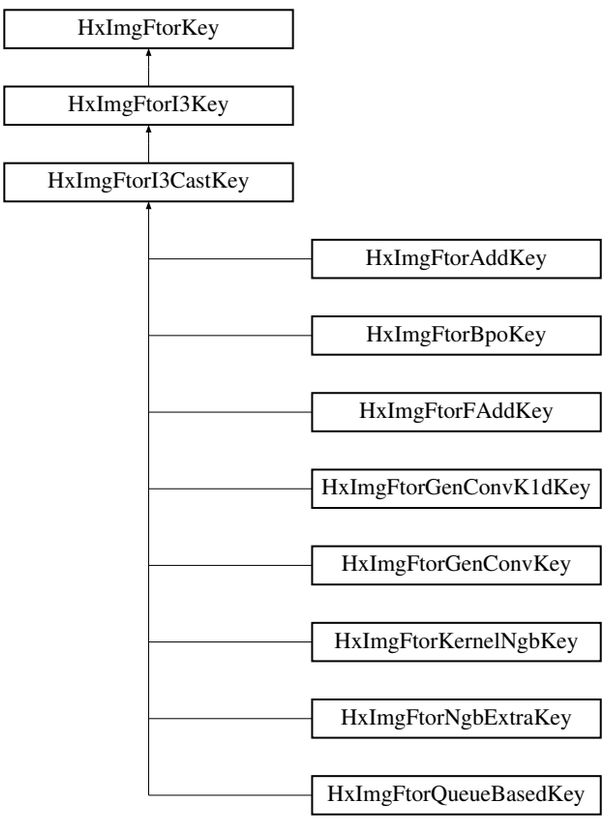
- **HxImgFtorI3Cast.h**
- HxImgFtorI3Cast.c

8.125 HxImgFtorI3CastKey Class Reference

Key for **HxImgFtorI3Cast** (p. 759).

`#include <HxImgFtorI3CastKey.h>`

Inheritance diagram for HxImgFtorI3CastKey::



Public Methods

- **HxImgFtorI3CastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name, **HxString** sig3Name)

  Constructor.

8.125.1 Detailed Description

Key for **HxImgFtorI3Cast** (p. 759).

8.125.2 Constructor & Destructor Documentation

8.125.2.1 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:

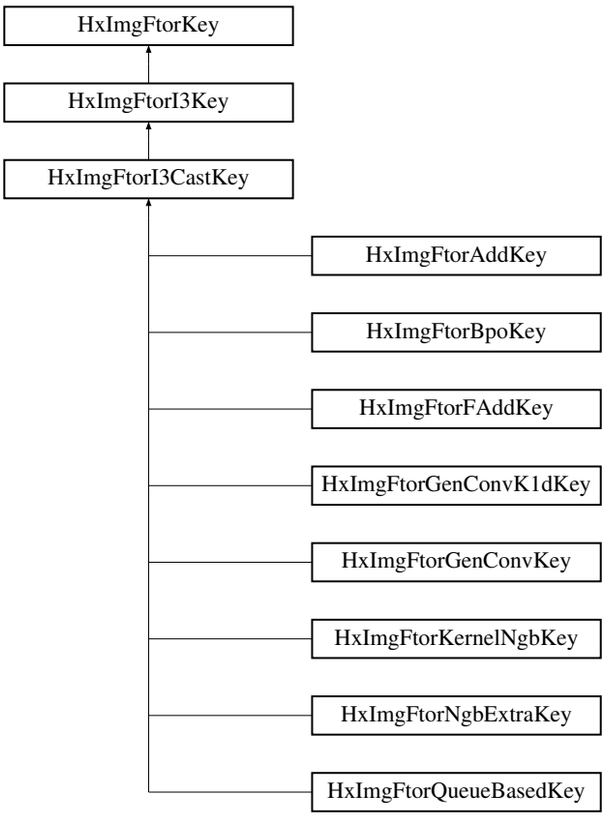
- **HxImgFtorI3CastKey.h**

8.126 HxImgFtorI3Key Class Reference

Key for **HxImgFtorI3** (p. 756).

`#include <HxImgFtorI3Key.h>`

Inheritance diagram for HxImgFtorI3Key::



Public Methods

- **HxImgFtorI3Key** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name, **HxString** sig3Name)

  Constructor.

8.126.1 Detailed Description

Key for **HxImgFtorI3** (p. 756).

8.126.2 Constructor & Destructor Documentation

8.126.2.1 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:

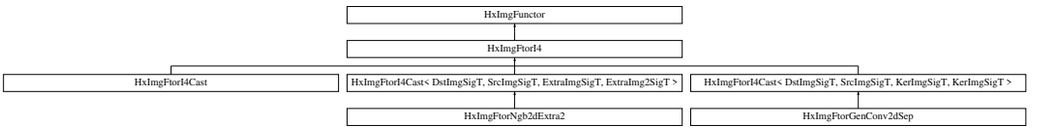
- **HxImgFtorI3Key.h**

8.127 HxImgFtorI4 Class Reference

Base class for image functors with four image parameters.

`#include <HxImgFtorI4.h>`

Inheritance diagram for HxImgFtorI4::



Public Types

- typedef **HxImgFtorI4Key KeyType**
  The key type of this class.

Public Methods

- **HxImgFtorI4** (const **KeyType** &)
  Constructor.
- virtual ~**HxImgFtorI4** ()
  Destructor.
- virtual void **callIt** (**HxImageData** *img1, **HxImageData** *img2, **HxImageData** *img3, **HxImageData** *img4, **HxTagList** &tags)=0
  callIt is implemented by **HxImgFtorI4Cast::callIt** (p. 775).

8.127.1 Detailed Description

Base class for image functors with four image parameters.

8.127.2 Member Typedef Documentation

8.127.2.1 typedef HxImgFtorI4Key HxImgFtorI4::KeyType

The key type of this class.

Reimplemented in **HxImgFtorGenConv2dSep** (p. 718), **HxImgFtorI4Cast** (p. 773), **HxImgFtorNgb2dExtra2** (p. 817), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorI4Cast< ImgSigT, KerSigT, KerSigT, KerSigT >** (p. 773), **HxImgFtorI4Cast< ResSigT, ImgSigT, ImgSigT, ImgSigT >** (p. 773), **HxImgFtorI4Cast< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT >** (p. 773), **HxImgFtorI4Cast< DstImgSigT, SrcImgSigT, KerImgSigT, KerImgSigT >** (p. 773), and **HxImgFtorNgb2dExtra2< ResSigT, ImgSigT, ImgSigT, ImgSigT, HxNgbOpticalFlow< typename ResSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 817).

8.127.3 Constructor & Destructor Documentation

8.127.3.1 HxImgFtorI4::HxImgFtorI4 (const KeyType & *key*) [inline]

Constructor.

```
47                                              : HxImgFunctor(key)
48 {
49 }
```

8.127.3.2 HxImgFtorI4::~HxImgFtorI4 () [inline, virtual]

Destructor.

```
53 {
54 }
```

8.127.4 Member Function Documentation

8.127.4.1 virtual void HxImgFtorI4::callIt (HxImageData * *img1*, HxImageData * *img2*, HxImageData * *img3*, HxImageData * *img4*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorI4Cast::callIt** (p. 775).

Reimplemented in **HxImgFtorI4Cast** (p. 775), **HxImgFtorI4Cast< ImgSigT, KerSigT, KerSigT, KerSigT >** (p. 775), **HxImgFtorI4Cast< ResSigT, ImgSigT, ImgSigT, ImgSigT >** (p. 775), **HxImgFtorI4Cast< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT >** (p. 775), and **HxImgFtorI4Cast< DstImgSigT, SrcImgSigT, KerImgSigT, KerImgSigT >** (p. 775).

The documentation for this class was generated from the following file:

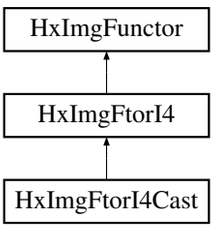
- **HxImgFtorI4.h**

8.128 HxImgFtorI4Cast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI4** (p. 770) to (statically typed) image data pointers.

`#include <HxImgFtorI4Cast.h>`

Inheritance diagram for HxImgFtorI4Cast::



Public Types

- typedef **HxImgFtorI4CastKey KeyType**

  The key type of this class.

- typedef Img1SigT::DataPtrType **Img1DataPtrType**

  The data pointer type of the first image.

- typedef Img2SigT::DataPtrType **Img2DataPtrType**

  The data pointer type of the second image.

- typedef Img3SigT::DataPtrType **Img3DataPtrType**

  The data pointer type of the third image.

- typedef Img4SigT::DataPtrType **Img4DataPtrType**

  The data pointer type of the fourth image.

Public Methods

- **HxImgFtorI4Cast** (const **KeyType** &)

  Constructor.

- virtual ~**HxImgFtorI4Cast** ()

  Destructor.

- virtual void **callIt** (**HxImageData** *img1, **HxImageData** *img2, **HxImageData** *img3, **HxImageData** *img4, **HxTagList** &tags)

Converts parameters and calls doIt.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** img1Ptr, **Img2DataPtrType** img2Ptr, **Img3DataPtrType** img3Ptr, **Img4DataPtrType** img4Ptr, **HxSizes** img1Size, **HxSizes** img2Size, **HxSizes** img3Size, **HxSizes** img4Size, **HxTagList** &tags, **HxImgFtorDescription** *=0)=0

  doIt is implemented by derived image functors:.

8.128.1 Detailed Description

template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> class HxImgFtorI4Cast< Img1SigT, Img2SigT, Img3SigT, Img4SigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorI4** (p. 770) to (statically typed) image data pointers.

Template parameters:

- Img1SigT is the signature type of the first image
- Img2SigT is the signature type of the second image
- Img3SigT is the signature type of the third image
- Img4SigT is the signature type of the fourth image

8.128.2 Member Typedef Documentation

8.128.2.1 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> typedef HxImgFtorI4CastKey HxImgFtorI4Cast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI4** (p. 770).

Reimplemented in **HxImgFtorGenConv2dSep** (p. 718), **HxImgFtorNgb2dExtra2** (p. 817), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), **HxImgFtorGenConv2dSep< 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. 718), and **HxImgFtorNgb2dExtra2< ResSigT, ImgSigT, ImgSigT, ImgSigT, HxNgbOpticalFlow< typename ResSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 817).

8.128.2.2 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> typedef Img1SigT::DataPtrType HxImgFtorI4Cast::Img1DataPtrType

The data pointer type of the first image.

8.128.2.3 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> typedef Img2SigT::DataPtrType HxImgFtorI4Cast::Img2DataPtrType

The data pointer type of the second image.

8.128.2.4 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> typedef Img3SigT::DataPtrType HxImgFtorI4Cast::Img3DataPtrType

The data pointer type of the third image.

8.128.2.5 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> typedef Img4SigT::DataPtrType HxImgFtorI4Cast::Img4DataPtrType

The data pointer type of the fourth image.

8.128.3 Constructor & Destructor Documentation

8.128.3.1 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> HxImgFtorI4Cast< Img1SigT, Img2SigT, Img3SigT, Img4SigT >::HxImgFtorI4Cast (const KeyType & *key*) [inline]

Constructor.

```
84                                          : HxImgFtorI4(key)
85 {
86 }
```

8.128.3.2 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> HxImgFtorI4Cast< Img1SigT, Img2SigT, Img3SigT, Img4SigT >::~HxImgFtorI4Cast () [virtual]

Destructor.

```
23 {
24 #ifdef CD_TRACE
25     HxEnvironment::instance()->outputStream()
26         << "~HxImgFtorI4Cast()" << STD_ENDL;
27     HxEnvironment::instance()->flush();
28 #endif
29 }
```

8.128.4 Member Function Documentation

8.128.4.1 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> void HxImgFtorI4Cast< Img1SigT, Img2SigT, Img3SigT, Img4SigT >::callIt (HxImageData * *img1*, HxImageData * *img2*, HxImageData * *img3*, HxImageData * *img4*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorI4** (p. 771).

```
36 {
37     TYPENAME Img1SigT::DataPtrType img1Ptr
38         = HxMakeDataPtr<typename Img1SigT::DataPtrType>(img1);
39     TYPENAME Img2SigT::DataPtrType img2Ptr
40         = HxMakeDataPtr<typename Img2SigT::DataPtrType>(img2);
41     TYPENAME Img3SigT::DataPtrType img3Ptr
42         = HxMakeDataPtr<typename Img3SigT::DataPtrType>(img3);
43     TYPENAME Img4SigT::DataPtrType img4Ptr
44         = HxMakeDataPtr<typename Img4SigT::DataPtrType>(img4);
45
46     HxImgFtorDescription* description = getDescription();
47     if (description)
48     {
49         description->setTags(tags);
50         description->addArgument(img1->signature(), img1->sizes());
51         description->addArgument(img2->signature(), img2->sizes());
52         description->addArgument(img3->signature(), img3->sizes());
53         description->addArgument(img4->signature(), img4->sizes());
54         description->startTime();
55     }
56
57     doIt(img1Ptr, img2Ptr, img3Ptr, img4Ptr,
58         img1->sizes(), img2->sizes(), img3->sizes(), img4->sizes(),
59         tags, description);
60
61     if (description)
62         description->stopTime();
63 }
```

8.128.4.2 template<class Img1SigT, class Img2SigT, class Img3SigT, class Img4SigT> virtual void HxImgFtorI4Cast< Img1SigT, Img2SigT, Img3SigT, Img4SigT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *extraPtr*, Img4DataPtrType *extra2Ptr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *extraSize*, HxSizes *extra2Size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, pure virtual]

doIt is implemented by derived image functors:.

- **HxImgFtorGenConv2dSep::doIt** (p. 719)

Reimplemented in **HxImgFtorGenConv2dSep** (p. 719), **HxImgFtorNgb2dExtra2** (p. 819), **HxImgFtorGenConv2dSep< 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. 719), **HxImgFtorGenConv2dSep< ImgSigT, KerSigT, KerSigT, HxBpoAdd< typename KerSigT::ArithType, typename KerSigT::ArithType, typename Ker-**

SigT::ArithType >, HxBpoMaxAssign< typename KerSigT::ArithType, typename KerSigT::ArithType >, HxKernel1d< typename KerSigT::DataPtrType, typename KerSigT::ArithType > > (p. 719), **HxImgFtorGenConv2dSep< 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. 719), **HxImgFtorGenConv2dSep< 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. 719), **HxImgFtorGenConv2dSep< 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. 719), and **HxImgFtorNgb2dExtra2< ResSigT, ImgSigT, ImgSigT, ImgSigT, HxNgbOpticalFlow< typename ResSigT::ArithTypeDouble, typename ImgSigT::ArithType > >** (p. 819).

The documentation for this class was generated from the following files:

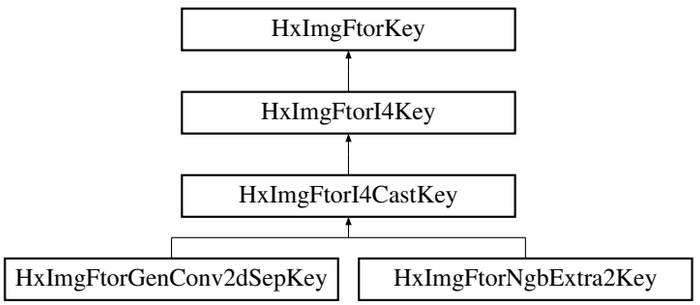
- **HxImgFtorI4Cast.h**
- HxImgFtorI4Cast.c

8.129 HxImgFtorI4CastKey Class Reference

Key for **HxImgFtorI4Cast** (p. 772).

`#include <HxImgFtorI4CastKey.h>`

Inheritance diagram for HxImgFtorI4CastKey::



Public Methods

- **HxImgFtorI4CastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name, **HxString** sig3Name, **HxString** sig4Name)

  Constructor.

8.129.1 Detailed Description

Key for **HxImgFtorI4Cast** (p. 772).

8.129.2 Constructor & Destructor Documentation

8.129.2.1 HxImgFtorI4CastKey::HxImgFtorI4CastKey (HxString *className*, HxString *sig1Name*, HxString *sig2Name*, HxString *sig3Name*, HxString *sig4Name*) [inline]

Constructor.

```
30     : HxImgFtorI4Key(className, sig1Name, sig2Name, sig3Name, sig4Name)
31 {
32 }
```

The documentation for this class was generated from the following file:

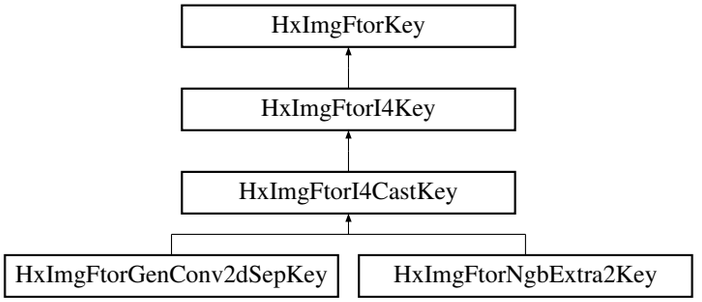
- **HxImgFtorI4CastKey.h**

8.130 HxImgFtorI4Key Class Reference

Key for **HxImgFtorI4** (p. 770).

`#include <HxImgFtorI4Key.h>`

Inheritance diagram for HxImgFtorI4Key::



Public Methods

- **HxImgFtorI4Key** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name, **HxString** sig3Name, **HxString** sig4Name)

  Constructor.

8.130.1 Detailed Description

Key for **HxImgFtorI4** (p. 770).

8.130.2 Constructor & Destructor Documentation

8.130.2.1 HxImgFtorI4Key::HxImgFtorI4Key (HxString *className*, HxString *sig1Name*, HxString *sig2Name*, HxString *sig3Name*, HxString *sig4Name*) [inline]

Constructor.

```
30     : HxImgFtorKey(className)
31 {
32     HxStringList l;
33     l << sig1Name << sig2Name << sig3Name << sig4Name;
34     setArguments(l.begin(),l.end());
35 }
```

The documentation for this class was generated from the following file:

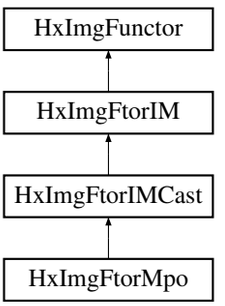
- **HxImgFtorI4Key.h**

8.131 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, int nImgs, **HxTagList** &tags)=0

  callIt is implemented by **HxImgFtorIMCast::callIt** (p. 782).

8.131.1 Detailed Description

Base class for image functors with 1 + M image parameters.

8.131.2 Member Typedef Documentation

8.131.2.1 typedef HxImgFtorIMKey HxImgFtorIM::KeyType

The key type of this class.

Reimplemented in **HxImgFtorIMCast** (p. 781), **HxImgFtorMpo** (p. 808), **HxImgFtorIMCast< DstSigT, SrcsSigT >** (p. 781), and **HxImgFtorMpo< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > >** (p. 808).

8.131.3 Constructor & Destructor Documentation

8.131.3.1 HxImgFtorIM::HxImgFtorIM (const KeyType & *key*) [inline]

Constructor.

```
49                                          : HxImgFunctor(key)
50 {
51 }
```

8.131.3.2 HxImgFtorIM::~HxImgFtorIM () [inline, virtual]

Destructor.

```
55 {
56 }
```

8.131.4 Member Function Documentation

8.131.4.1 virtual void HxImgFtorIM::callIt (HxImageData * *dstImg*, HxImageData ** *srcImgs*, int *nImgs*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorIMCast::callIt** (p. 782).

Reimplemented in **HxImgFtorIMCast** (p. 782), and **HxImgFtorIMCast< DstSigT, SrcsSigT >** (p. 782).

The documentation for this class was generated from the following file:

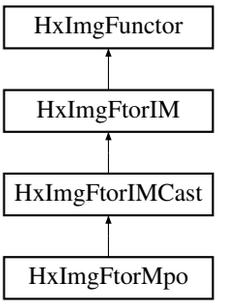
- **HxImgFtorIM.h**

8.132 HxImgFtorIMCast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorIM** (p. 778) 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, 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:.

8.132.1 Detailed Description

template<class DstImgSigT, class SrcImgsSigT> class HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorIM** (p. 778) to (statically typed) image data pointers.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgsSigT is the signature type of the source images

8.132.2 Member Typedef Documentation

8.132.2.1 template<class DstImgSigT, class SrcImgsSigT> typedef HxImgFtorIMCastKey HxImgFtorIMCast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIM** (p. 779).

Reimplemented in **HxImgFtorMpo** (p. 808), and **HxImgFtorMpo< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > >** (p. 808).

8.132.2.2 template<class DstImgSigT, class SrcImgsSigT> typedef DstImgSigT::DataPtrType HxImgFtorIMCast::DstDataPtrType

The data pointer type of the destination image.

8.132.2.3 template<class DstImgSigT, class SrcImgsSigT> typedef SrcImgsSigT::DataPtrType HxImgFtorIMCast::SrcDataPtrType

The data pointer type of the source images.

8.132.2.4 template<class DstImgSigT, class SrcImgsSigT> typedef HxDataPtrArray<SrcImgsSigT> HxImgFtorIMCast::SrcDataPtrArray

An array of data pointers to source images.

8.132.3 Constructor & Destructor Documentation

8.132.3.1 template<class DstImgSigT, class SrcImgsSigT> HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >::HxImgFtorIMCast (const KeyType & *key*) [inline]

Constructor.

```
81             : HxImgFtorIM(key){}
```

8.132.3.2 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 }
```

8.132.4 Member Function Documentation

8.132.4.1 template<class DstImgSigT, class SrcImgsSigT> void HxImgFtorIMCast< DstImgSigT, SrcImgsSigT >::callIt (HxImageData * *dstImg*, HxImageData ** *srcImgs*, int *nImgs*, HxTagList & *tags*) [virtual]

Converts parameters and calls doIt.

Reimplemented from **HxImgFtorIM** (p. 779).

```
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 }
```

8.132.4.2 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:.

- **HxImgFtorMpo::doIt** (p. 809)

Reimplemented in **HxImgFtorMpo** (p. 809), and **HxImgFtorMpo< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > >** (p. 809).

The documentation for this class was generated from the following files:

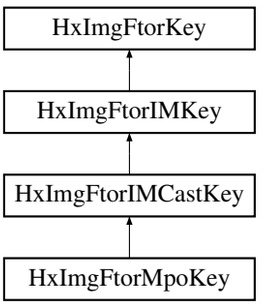
- **HxImgFtorIMCast.h**
- HxImgFtorIMCast.c

8.133 HxImgFtorIMCastKey Class Reference

Key for **HxImgFtorIMCast** (p. 779).

`#include <HxImgFtorIMCastKey.h>`

Inheritance diagram for HxImgFtorIMCastKey::



Public Methods

- **HxImgFtorIMCastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

8.133.1 Detailed Description

Key for **HxImgFtorIMCast** (p. 779).

8.133.2 Constructor & Destructor Documentation

8.133.2.1 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:

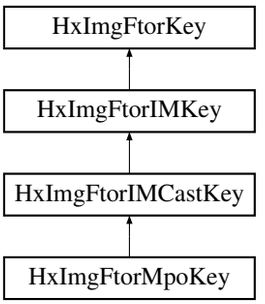
- **HxImgFtorIMCastKey.h**

8.134 HxImgFtorIMKey Class Reference

Key for **HxImgFtorIM** (p. 778).

`#include <HxImgFtorIMKey.h>`

Inheritance diagram for HxImgFtorIMKey::



Public Methods

- **HxImgFtorIMKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

8.134.1 Detailed Description

Key for **HxImgFtorIM** (p. 778).

8.134.2 Constructor & Destructor Documentation

8.134.2.1 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:

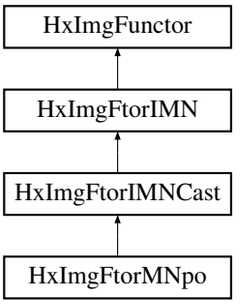
- **HxImgFtorIMKey.h**

8.135 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::callIt** (p. 789).

8.135.1 Detailed Description

Base class for image functors with M + N image parameters.

8.135.2 Member Typedef Documentation

8.135.2.1 typedef HxImgFtorIMNKey HxImgFtorIMN::KeyType

The key type of this class.

Reimplemented in **HxImgFtorIMNCast** (p. 788), and **HxImgFtorMNpo** (p. 805).

8.135.3 Constructor & Destructor Documentation

8.135.3.1 HxImgFtorIMN::HxImgFtorIMN (const KeyType & *key*) [inline]

Constructor.

```
50                                              : HxImgFunctor(key)
51 {
52 }
```

8.135.3.2 HxImgFtorIMN::~HxImgFtorIMN () [inline, virtual]

Destructor.

```
56 {
57 }
```

8.135.4 Member Function Documentation

8.135.4.1 virtual void HxImgFtorIMN::callIt (HxImageData ** *dstImgs*, int *dstCnt*, HxImageData ** *srcImgs*, int *srcCnt*, HxTagList & *tags*) [pure virtual]

callIt is implemented by **HxImgFtorIMNCast::callIt** (p. 789).

Reimplemented in **HxImgFtorIMNCast** (p. 789).

The documentation for this class was generated from the following file:

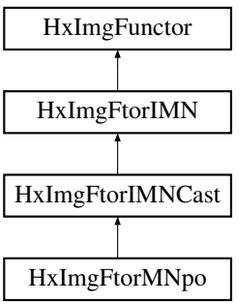
- **HxImgFtorIMN.h**

8.136 HxImgFtorIMNCast Class Template Reference

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorIMN** (p. 784) 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:.

8.136.1 Detailed Description

template<class DstImgsSigT, class SrcImgsSigT> class HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >

Class for (checked) conversion of (polymorphic) **HxImageData** (p. 581) parameters of **HxImgFtorIMN** (p. 784) to (statically typed) image data pointers.

Template parameters:

- DstImgsSigT is the signature type of the destination images
- SrcImgsSigT is the signature type of the source images

8.136.2 Member Typedef Documentation

8.136.2.1 template<class DstImgsSigT, class SrcImgsSigT> typedef HxImgFtorIMNCastKey HxImgFtorIMNCast::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIMN** (p. 785).

Reimplemented in **HxImgFtorMNpo** (p. 805).

8.136.2.2 template<class DstImgsSigT, class SrcImgsSigT> typedef DstImgsSigT::DataPtrType HxImgFtorIMNCast::DstDataPtrType

The data pointer type of the destination images.

8.136.2.3 template<class DstImgsSigT, class SrcImgsSigT> typedef SrcImgsSigT::DataPtrType HxImgFtorIMNCast::SrcDataPtrType

The data pointer type of the source images.

8.136.2.4 template<class DstImgsSigT, class SrcImgsSigT> typedef HxDataPtrArray<SrcImgsSigT> HxImgFtorIMNCast::SrcDataPtrArray

An array of data pointers to source images.

8.136.2.5 template<class DstImgsSigT, class SrcImgsSigT> typedef HxDataPtrArray<DstImgsSigT> HxImgFtorIMNCast::DstDataPtrArray

An array of data pointers to destination images.

8.136.3 Constructor & Destructor Documentation

8.136.3.1 template<class DstImgsSigT, class SrcImgsSigT> HxImgFtorIMNCast< DstImgsSigT, SrcImgsSigT >::HxImgFtorIMNCast (const KeyType & *key*) [inline]

Constructor.

```
84             : HxImgFtorIMN(key){}
```

8.136.3.2 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 }
```

8.136.4 Member Function Documentation

8.136.4.1 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. 786).

```
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 }
```

8.136.4.2 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:.

- **HxImgFtorMNpo::doIt** (p. 806)

Reimplemented in **HxImgFtorMNpo** (p. 806).

The documentation for this class was generated from the following files:

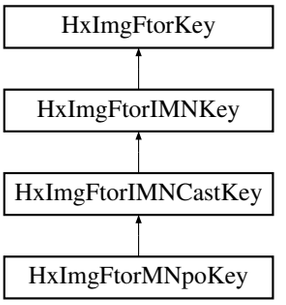
- **HxImgFtorIMNCast.h**
- HxImgFtorIMNCast.c

8.137 HxImgFtorIMNCastKey Class Reference

Key for **HxImgFtorIMNCast** (p. 786).

`#include <HxImgFtorIMNCastKey.h>`

Inheritance diagram for HxImgFtorIMNCastKey::



Public Methods

- **HxImgFtorIMNCastKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

8.137.1 Detailed Description

Key for **HxImgFtorIMNCast** (p. 786).

8.137.2 Constructor & Destructor Documentation

8.137.2.1 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:

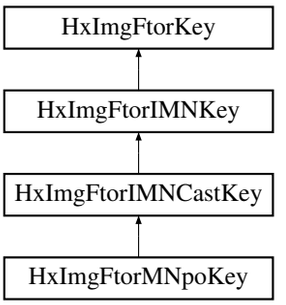
- **HxImgFtorIMNCastKey.h**

8.138 HxImgFtorIMNKey Class Reference

Key for **HxImgFtorIMN** (p. 784).

`#include <HxImgFtorIMNKey.h>`

Inheritance diagram for HxImgFtorIMNKey::



Public Methods

- **HxImgFtorIMNKey** (**HxString** className, **HxString** sig1Name, **HxString** sig2Name)
  Constructor.

8.138.1 Detailed Description

Key for **HxImgFtorIMN** (p. 784).

8.138.2 Constructor & Destructor Documentation

8.138.2.1 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:

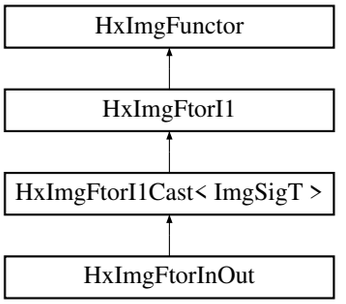
- **HxImgFtorIMNKey.h**

8.139 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 **HxFuncInOutInit** (p. 193) *to do the initialization phase and* **HxFuncInOutDispatch** (p. 190) *to dispatch the actual work.*

8.139.1 Detailed Description

template<class ImgSigT, class InOutT> class HxImgFtorInOut< ImgSigT, InOutT >

Instantiation of generic algorithm for in/out pixel operations on images.

Template parameters:

- ImgSigT is the signature type of the image
- InOutT is the type of the in/out pixel functor

8.139.2 Member Typedef Documentation

8.139.2.1 template<class ImgSigT, class InOutT> typedef HxImgFtorInOutKey HxImgFtorInOut::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 732).

8.139.3 Constructor & Destructor Documentation

8.139.3.1 template<class ImgSigT, class InOutT> HxImgFtorInOut< ImgSigT, InOutT >::HxImgFtorInOut () [inline]

Constructor.

```
36     : HxImgFtorI1Cast<ImgSigT>(
37         HxImgFtorInOutKey(HxClassName<ImgSigT>(), HxClassName<InOutT>()))
38 {
39     HxImgFtorRuleBase::instance().setIsModifying(
40         "inout", HxClassName<InOutT>(),
41         IsModifying(typename InOutT::DirectionCategory()));
42 }
```

8.139.3.2 template<class ImgSigT, class InOutT> HxImgFtorInOut< ImgSigT, InOutT >::~HxImgFtorInOut () [virtual]

Destructor.

```
46 {
47 }
```

8.139.4 Member Function Documentation

8.139.4.1 template<class ImgSigT, class InOutT> void HxImgFtorInOut< ImgSigT, InOutT >::doIt (ImgDataPtrType *ptr*, HxSizes *size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncInOutInit** (p. 193) to do the initialization phase and **HxFuncInOutDispatch** (p. 190) to dispatch the actual work.

Reimplemented from **HxImgFtorI1Cast** (p. 734).

```
54 {
55     if (description) {
56         HxString v(typename InOutT::DirectionCategory().toString());
57         v += ", ";
58         v += typename InOutT::TransVarianceCategory().toString();
59         v += ", ";
60         v += typename InOutT::PhaseCategory().toString();
61         description->setVariation(v);
62     }
63
64     HxBoundingBox imgBb(size), regionBb(size);
```

```
65      regionBb = HxGetTag(tags, "boundingBox", imgBb);
66      regionBb = regionBb.intersect(imgBb);
67
68      if (!regionBb.isEmpty())
69      {
70          InOutT pixOp =
71              HxFuncInOutInit<InOutT>(
72                  regionBb.size(), tags,
73                  typename InOutT::DirectionCategory(),
74                  typename InOutT::TransVarianceCategory(),
75                  typename InOutT::PhaseCategory());
76          ptr.incXYZ(
77              regionBb.begin().x(), regionBb.begin().y(), regionBb.begin().z());
78          HxFuncInOutDispatch(ptr, regionBb.size(), pixOp);
79      }
80
81  }
```

The documentation for this class was generated from the following files:

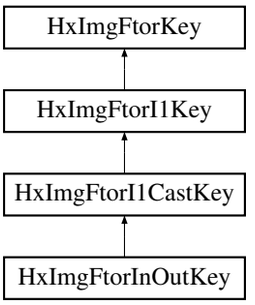
- **HxImgFtorInOut.h**
- HxImgFtorInOut.c

8.140 HxImgFtorInOutKey Class Reference

Key for **HxImgFtorInOut** (p. 791).

`#include <HxImgFtorInOutKey.h>`

Inheritance diagram for HxImgFtorInOutKey::



Public Methods

- **HxImgFtorInOutKey** (**HxString** imgSig, **HxString** inOutName)

  Constructor.

8.140.1 Detailed Description

Key for **HxImgFtorInOut** (p. 791).

8.140.2 Constructor & Destructor Documentation

8.140.2.1 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:

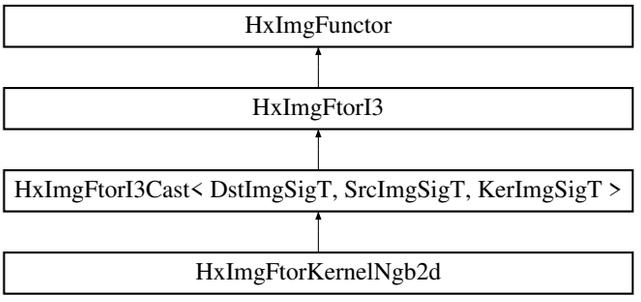
- **HxImgFtorInOutKey.h**

8.141 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.

Public Methods

- **HxImgFtorKernelNgb2d** ()
  Constructor.
- virtual ~**HxImgFtorKernelNgb2d** ()
  Destructor.
- virtual bool **probeOp** (**HxTagList** &tags) const
  Probe for border size.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** kerPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.141.1 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.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- KerImgSigT is the signature type of the kernel image
- NgbT is the type of the neighbourhood functor

8.141.2 Member Typedef Documentation

8.141.2.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> typedef HxImgFtorKernelNgbKey HxImgFtorKernelNgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 761).

8.141.3 Constructor & Destructor Documentation

8.141.3.1 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 }
```

8.141.3.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::~HxImgFtorKernelNgb2d () [virtual]

Destructor.

```
39 {
40 }
```

8.141.4 Member Function Documentation

8.141.4.1 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> bool HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::probeOp (HxTagList & *tags*) const [virtual]

Probe for border size.

Reimplemented from **HxImgFunctor** (p. 864).

```
46 {
47     NgbT ngb(tags);
48     HxSizes borderSize = ngb.size() / HxSizes(2,2,2);
49     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
50     return true;
51 }
```

8.141.4.2 template<class DstImgSigT, class SrcImgSigT, class KerImgSigT, class NgbT> void HxImgFtorKernelNgb2d< DstImgSigT, SrcImgSigT, KerImgSigT, NgbT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *kerPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr	Output image: IS = dstSize, IBS = 0
srcPtr	Input image: IS = srcSize, IBS = taglist(borderSize)
kerPtr	Input image, IS = kerSize, IBS = 0

Calls **HxFuncKernelNgbOp2dDispatch** (p. 196) to dispatch the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
62 {
63     typedef HxKernel2d<Img3DataPtrType, typename KerImgSigT::ArithType> KernelType;
64
65     NgbT        ngb(tags);
66     KernelType  kernel(kerPtr, kerSize, tags);
67
68     if (description) {
69         HxString v(typename NgbT::IteratorCategory().toString());
70         v += ", ";
71         v += typename NgbT::PhaseCategory().toString();
72         description->setVariation(v);
```

```
73     }
74
75     if (kerSize.inf(ngb.size()) != kerSize) {
76         HxEnvironment::instance()->errorStream()
77             << "HxImgFtorKernelNgb2d: kernel larger than neighbourhood"
78             << STD_ENDL;
79         HxEnvironment::instance()->flush();
80         return;
81     }
82
83     HxFuncKernelNgbOp2dDispatch(dstPtr, srcPtr, dstSize, ngb, kernel);
84 }
```

The documentation for this class was generated from the following files:

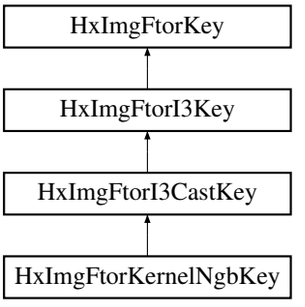
- **HxImgFtorKernelNgb2d.h**
- HxImgFtorKernelNgb2d.c

8.142 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.

8.142.1 Detailed Description

Key for HxImgFtorKernelNgb.

8.142.2 Constructor & Destructor Documentation

8.142.2.1 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:

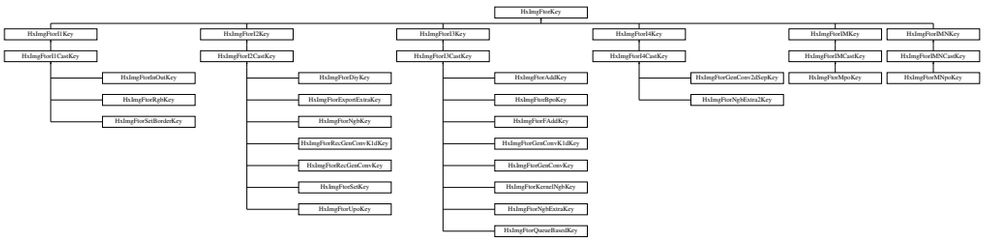
- **HxImgFtorKernelNgbKey.h**

8.143 HxImgFtorKey Class Reference

Base class for keys for image functors.

`#include <HxImgFtorKey.h>`

Inheritance diagram for HxImgFtorKey::



Public Methods

- **HxImgFtorKey** ()

  Constructor.

- **HxImgFtorKey** (const HxImgFtorKey &)

  Copy constructor.

- **HxImgFtorKey** (**HxString** className, **HxStringListIter** argListBegin, **HxStringListIter** argListEnd)

  Constructor.

- **HxImgFtorKey** (**HxString** className)

  Constructor.

- void **addArgument** (**HxString** argName)

  Add argument.

- void **setArguments** (**HxStringListIter** argListBegin, **HxStringListIter** argListEnd)

  Set arguments.

- **HxString getArgument** (int) const

  Get argument.

- **HxString getClassName** () const

  Get class name.

- **HxString toString** () const

  Convert key to string.

- int **compare** (const HxImgFtorKey &) const

  Compare keys.

- STD_OSTREAM & **put** (STD_OSTREAM &) const

  Put on given stream.

8.143.1 Detailed Description

Base class for keys for image functors.

A key is a list of strings where the first string corresponds to the name of the image functor and all other (existing) strings correspond to the template parameters of the functor class.

8.143.2 Constructor & Destructor Documentation

8.143.2.1 HxImgFtorKey::HxImgFtorKey ()

Constructor.

```
15 {
16     for(int i = 0;i<maxKeySize;i++)
17         _key[i] = 0;
18 }
```

8.143.2.2 HxImgFtorKey::HxImgFtorKey (const HxImgFtorKey & *arg*)

Copy constructor.

```
21 {
22     for(int i = 0;i<maxKeySize;i++)
23         _key[i] = arg._key[i];
24 }
```

8.143.2.3 HxImgFtorKey::HxImgFtorKey (HxString *className*, HxStringListIter *argListBegin*, HxStringListIter *argListEnd*)

Constructor.

```
28 {
29     for(int i = 0;i<maxKeySize;i++)
30         _key[i] = 0;
31     _key[0] = uShort(nameTable().getClassNameId(className));
32     setArguments(argListBegin, argListEnd);
33 }
```

8.143.2.4 HxImgFtorKey::HxImgFtorKey (HxString *className*)

Constructor.

```
36 {
37     for(int i = 0;i<maxKeySize;i++)
38         _key[i] = 0;
39     _key[0] = uShort(nameTable().getClassNameId(className));
40 }
```

8.143.3 Member Function Documentation

8.143.3.1 void HxImgFtorKey::addArgument (HxString *argName*)

Add argument.

```
44 {
45     int i;
46     for (i=1; i<maxKeySize; i++)
47         if (!_key[i]) break;
48     if ((!_key[i]) && (i<maxKeySize-1))
49     {
50         _key[i] = uShort(nameTable().getTypeNameId(argName));
51         _key[i+1] = 0;
52     }
53 }
```

8.143.3.2 void HxImgFtorKey::setArguments (HxStringListIter *argListBegin*, HxStringListIter *argListEnd*)

Set arguments.

```
59 {
60     int i;
61     for(    i = 1;
62             (argListBegin != argListEnd) && (i < maxKeySize - 1);
63             i++, argListBegin++)
64         _key[i] = uShort(nameTable().getTypeNameId(*argListBegin));
65     _key[i] = 0;
66 }
```

8.143.3.3 HxString HxImgFtorKey::getArgument (int *i*) const

Get argument.

```
70 {
71     return ((i < maxKeySize-1) && _key[i+1]) ?
72         nameTable().getTypeName(_key[i+1]) : HxString("");
73 }
```

8.143.3.4 HxString HxImgFtorKey::getClassName () const

Get class name.

```
77 {
78     return nameTable().getClassName(_key[0]);
79 }
```

8.143.3.5 HxString HxImgFtorKey::toString () const

Convert key to string.

```
83 {
84     HxString s(nameTable().getClassName(_key[0]));
85
86     int i;
87
88     for(i = 1;_key[i];i++)
89     {
90         if(i == 1)
91             s += "<";
92         else
93             s += ", ";
94         s += nameTable().getTypeName(_key[i]);
95     }
96
97     if(i > 1)
98         s += ">";
99
100     return(s);
101 }
```

8.143.3.6 int HxImgFtorKey::compare (const HxImgFtorKey & *arg*) const

Compare keys.

```
105 {
106     int i;
107     for(i=0;_key[i] && arg._key[i]; i++)
108         if(_key[i] != arg._key[i])
109             break;
110     return(_key[i] - arg._key[i]);
111 }
```

8.143.3.7 STD_OSTREAM & HxImgFtorKey::put (STD_OSTREAM & *os*) const

Put on given stream.

```
115 {
116     return os << toString();
117 }
```

The documentation for this class was generated from the following files:

- **HxImgFtorKey.h**
- HxImgFtorKey.c

8.144 HxImgFtorKeyNameTable Class Reference

Name table for class and type names that occur in image functor 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** ()

8.144.1 Detailed Description

Name table for class and type names that occur in image functor keys.

The documentation for this class was generated from the following files:

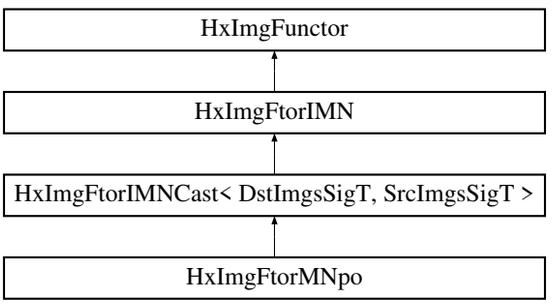
- **HxImgFtorKeyNameTable.h**
- HxImgFtorKeyNameTable.c

8.145 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** *description=0)

  Calls **HxFuncMNpoDispatch** (p. 197) *to do the actual work.*

8.145.1 Detailed Description

template<class DstImgsSigT, class SrcImgsSigT, class MNpoT> class HxImgFtorMNpo< DstImgsSigT, SrcImgsSigT, MNpoT >

Instantiation of generic algorithm for M output, N input pixel operations on images.

Template parameters:

- DstImgsSigT is the signature type of the destination images
- SrcImgsSigT is the signature type of the source images
- MNpoT is the type of the multi pixel functor

8.145.2 Member Typedef Documentation

8.145.2.1 template<class DstImgsSigT, class SrcImgsSigT, class MNpoT> typedef HxImgFtorMNpoKey HxImgFtorMNpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIMNCast** (p. 788).

8.145.3 Constructor & Destructor Documentation

8.145.3.1 template<class DstImgsSigT, class SrcImgsSigT, class MNpoT> HxImgFtorMNpo< DstImgsSigT, SrcImgsSigT, MNpoT >::HxImgFtorMNpo ()

Constructor.

```
21     : HxImgFtorIMNCast<DstImgsSigT, SrcImgsSigT>(
22         HxImgFtorMNpoKey(HxClassName<DstImgsSigT>(), HxClassName<SrcImgsSigT>(),
23             HxClassName<MNpoT>()))
24 {
25 #ifdef CD_TRACE
26     HxEnvironment::instance()->outputStream()
27         << "HxImgFtorMNpo::HxImgFtorMNpo()" << STD_ENDL;
28 #endif
29     HxImgFtorRuleBase::instance().setResultType(
30         HxClassName<DstImgsSigT>(), "mnpo",
31         HxClassName<SrcImgsSigT>(), HxClassName<MNpoT>());
32 }
```

8.145.3.2 template<class DstImgsSigT, class SrcImgsSigT, class MNpoT> HxImgFtorMNpo< DstImgsSigT, SrcImgsSigT, MNpoT >::~HxImgFtorMNpo () [virtual]

Destructor.

```
36 {
37 #ifdef CD_TRACE
38     HxEnvironment::instance()->outputStream()
39         << "HxImgFtorMNpo::~HxImgFtorMNpo()" << STD_ENDL;
40 #endif
41 }
```

8.145.4 Member Function Documentation

8.145.4.1 template<class DstImgsSigT, class SrcImgsSigT, class MNpoT> bool HxImgFtorMNpo< DstImgsSigT, SrcImgsSigT, MNpoT >::probeOp (HxTagList & *tags*) const [virtual]

Probe Operation.

Reimplemented from **HxImgFunctor** (p. 864).

```
46 {
47     MNpoT mpo(tags);
48
49     return HxGetTag(tags, "preOpIsOk", true);
50 }
```

8.145.4.2 template<class DstImgsSigT, class SrcImgsSigT, class MNpoT> void HxImgFtorMNpo< DstImgsSigT, SrcImgsSigT, MNpoT >::doIt (DstDataPtrArray & *dstPtrs*, SrcDataPtrArray & *srcPtrs*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncMNpoDispatch** (p. 197) to do the actual work.

Reimplemented from **HxImgFtorIMNCast** (p. 789).

```
58 {
59     HxAddTag(tags, "sourceCnt", srcPtrs.size());
60     MNpoT mpo(tags);
61
62     if (description) {
63         HxString v(typename MNpoT::TransVarianceCategory().toString());
64         description->setVariation(v);
65     }
66
67     HxFuncMNpoDispatch(dstPtrs, srcPtrs, dstSize, mpo);
68 }
```

The documentation for this class was generated from the following files:

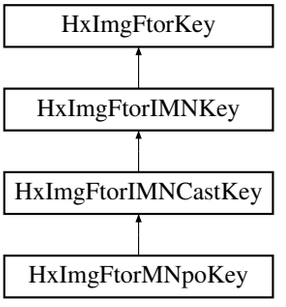
- **HxImgFtorMNpo.h**
- HxImgFtorMNpo.c

8.146 HxImgFtorMNpoKey Class Reference

Key for **HxImgFtorMNpo** (p. 804).

`#include <HxImgFtorMNpoKey.h>`

Inheritance diagram for HxImgFtorMNpoKey::



Public Methods

- **HxImgFtorMNpoKey** (**HxString** dstImgsSig, **HxString** srcImgsSig, **HxString** mpoName)
  Constructor.

8.146.1 Detailed Description

Key for **HxImgFtorMNpo** (p. 804).

8.146.2 Constructor & Destructor Documentation

8.146.2.1 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:

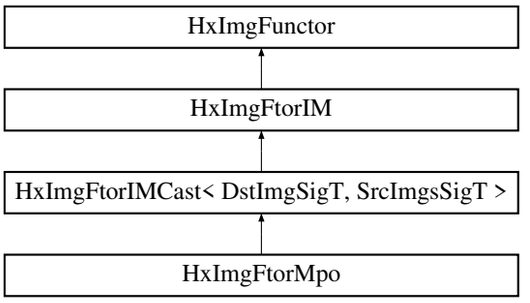
- **HxImgFtorMNpoKey.h**

8.147 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** ∗description=0)

  Calls **HxFuncMpoDispatch** (p. 198) *to do the actual work.*

8.147.1 Detailed Description

template<class DstImgSigT, class SrcImgsSigT, class MpoT> class HxImgFtorMpo< DstImgSigT, SrcImgsSigT, MpoT >

Instantiation of generic algorithm for multi pixel operations on images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgsSigT is the signature type of the source images
- MpoT is the type of the multi pixel functor

8.147.2 Member Typedef Documentation

8.147.2.1 template<class DstImgSigT, class SrcImgsSigT, class MpoT> typedef HxImgFtorMpoKey HxImgFtorMpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorIMCast** (p. 781).

8.147.3 Constructor & Destructor Documentation

8.147.3.1 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     HxImgFtorRuleBase::instance().setResultType(
31         HxClassName<DstImgSigT>(), "mpo",
32         HxClassName<SrcImgsSigT>(), HxClassName<MpoT>());
33 }
```

8.147.3.2 template<class DstImgSigT, class SrcImgsSigT, class MpoT> HxImgFtorMpo< DstImgSigT, SrcImgsSigT, MpoT >::~HxImgFtorMpo () [virtual]

Destructor.

```
37 {
38 #ifdef CD_TRACE
39     HxEnvironment::instance()->outputStream()
40         << "HxImgFtorMpo::~HxImgFtorMpo()" << STD_ENDL;
41 #endif
42 }
```

8.147.4 Member Function Documentation

8.147.4.1 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 **HxFuncMpoDispatch** (p. 198) to do the actual work.

Reimplemented from **HxImgFtorIMCast** (p. 782).

```
50 {
51     HxAddTag(tags, "sourceCnt", srcPtrs.size());
52     MpoT mpo(tags);
53
54     if (description) {
55         HxString v(typename MpoT::TransVarianceCategory().toString());
56         description->setVariation(v);
57     }
58
59     HxFuncMpoDispatch(dstPtr, srcPtrs, dstSize, mpo);
60 }
```

The documentation for this class was generated from the following files:

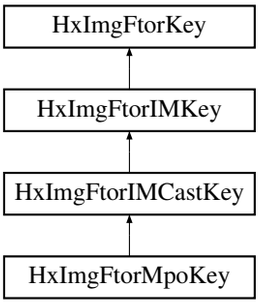
- **HxImgFtorMpo.h**
- HxImgFtorMpo.c

8.148 HxImgFtorMpoKey Class Reference

Key for **HxImgFtorMpo** (p. 807).

`#include <HxImgFtorMpoKey.h>`

Inheritance diagram for HxImgFtorMpoKey::



Public Methods

- **HxImgFtorMpoKey** (**HxString** dstImgSig, **HxString** srcImgsSig, **HxString** mpoName)
  Constructor.

8.148.1 Detailed Description

Key for **HxImgFtorMpo** (p. 807).

8.148.2 Constructor & Destructor Documentation

8.148.2.1 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:

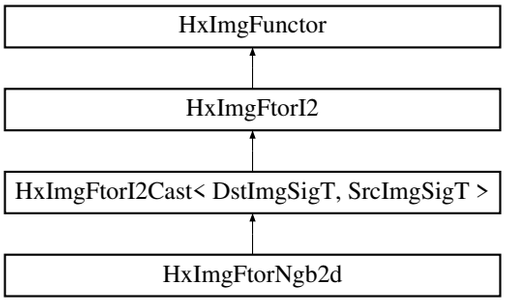
- **HxImgFtorMpoKey.h**

8.149 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 bool **probeOp** (**HxTagList** &tags) const

  Probe for border size.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Do it.

8.149.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class NgbT> class HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >

Instantiation of generic algorithm for neighborhood operations on 2D images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- NgbT is the type of the neighbourhood functor

8.149.2 Member Typedef Documentation

8.149.2.1 template<class DstImgSigT, class SrcImgSigT, class NgbT> typedef HxImgFtorNgbKey HxImgFtorNgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.149.3 Constructor & Destructor Documentation

8.149.3.1 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 }
```

8.149.3.2 template<class DstImgSigT, class SrcImgSigT, class NgbT> HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::~HxImgFtorNgb2d () [virtual]

Destructor.

```
34 {
35 }
```

8.149.4 Member Function Documentation

8.149.4.1 template<class DstImgSigT, class SrcImgSigT, class NgbT> bool HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::probeOp (HxTagList & *tags*) const [virtual]

Probe for border size.

Reimplemented from **HxImgFunctor** (p. 864).

```
41 {
42     NgbT ngb(tags);
43     HxSizes borderSize = ngb.size() / HxSizes(2,2,2);
44     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
45     return true;
46 }
```

8.149.4.2 template<class DstImgSigT, class SrcImgSigT, class NgbT> void HxImgFtorNgb2d< DstImgSigT, SrcImgSigT, NgbT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = taglist(borderSize)

Calls **HxFuncNgbOp2dDispatch** (p. 202) to dispatch the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
55 {
56     NgbT ngb(tags);
57
58     if (description) {
59         HxString v(typename NgbT::IteratorCategory().toString());
60         v += ", ";
61         v += typename NgbT::PhaseCategory().toString();
62         description->setVariation(v);
63     }
64
65     HxFuncNgbOp2dDispatch(dstPtr, srcPtr, dstSize, ngb);
66 }
```

The documentation for this class was generated from the following files:

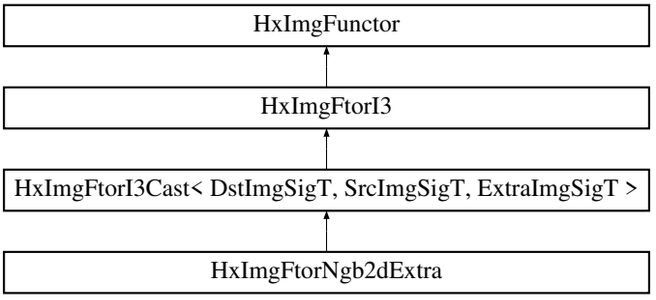
- **HxImgFtorNgb2d.h**
- HxImgFtorNgb2d.c

8.150 HxImgFtorNgb2dExtra Class Template Reference

Instantiation of generic algorithm for neighborhood operations on 2D images with an extra image.

`#include <HxImgFtorNgb2dExtra.h>`

Inheritance diagram for HxImgFtorNgb2dExtra::



Public Types

- typedef **HxImgFtorNgbExtraKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorNgb2dExtra** ()

  Constructor.

- virtual ∼**HxImgFtorNgb2dExtra** ()

  Destructor.

- virtual bool **probeOp** (**HxTagList** &tags) const

  Probe for border size.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** extraPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** extraSize, **HxTagList** &tags, **HxImgFtorDescription** ∗description=0)

  Do it.

8.150.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class NgbT> class HxImgFtorNgb2dExtra< DstImgSigT, SrcImgSigT, ExtraImgSigT, NgbT >

Instantiation of generic algorithm for neighborhood operations on 2D images with an extra image.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- ExtraImgSigT is the signature type of the extra image
- NgbT is the type of the neighbourhood functor

8.150.2 Member Typedef Documentation

8.150.2.1 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class NgbT> typedef HxImgFtorNgbExtraKey HxImgFtorNgb2dExtra::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI3Cast** (p. 761).

8.150.3 Constructor & Destructor Documentation

8.150.3.1 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class NgbT> HxImgFtorNgb2dExtra< DstImgSigT, SrcImgSigT, ExtraImgSigT, NgbT >::HxImgFtorNgb2dExtra () [inline]

Constructor.

```
20     : HxImgFtorI3Cast<DstImgSigT, SrcImgSigT, ExtraImgSigT>(
21         HxImgFtorNgbExtraKey(
22             HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
23             HxClassName<ExtraImgSigT>(), HxClassName<NgbT>()))
24 {
25     // Using ngb instead of something like ngbExtra...
26     HxImgFtorRuleBase::instance().setResultType(
27         HxClassName<DstImgSigT>(), "ngb",
28         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
29     HxImgFtorRuleBase::instance().setExtraType(
30         HxClassName<ExtraImgSigT>(), "ngb",
31         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
32 }
```

8.150.3.2 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class NgbT> HxImgFtorNgb2dExtra< DstImgSigT, SrcImgSigT, ExtraImgSigT, NgbT >::~HxImgFtorNgb2dExtra () [virtual]

Destructor.

```
37 {
38 }
```

8.150.4 Member Function Documentation

8.150.4.1 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class NgbT> bool HxImgFtorNgb2dExtra< DstImgSigT, SrcImgSigT, ExtraImgSigT, NgbT >::probeOp (HxTagList & *tags*) const [virtual]

Probe for border size.

Reimplemented from **HxImgFunctor** (p. 864).

```
44 {
45     NgbT ngb(tags);
46     HxSizes borderSize = ngb.size() / HxSizes(2,2,2);
47     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
48     return true;
49 }
```

8.150.4.2 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class NgbT> void HxImgFtorNgb2dExtra< DstImgSigT, SrcImgSigT, ExtraImgSigT, NgbT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *extraPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *extraSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = taglist(borderSize)

extraPtr Extra image: IS = extraSize, IBS = taglist(borderSize)

Calls **HxFuncNgbOp2dExtraDispatch** (p. 206) to dispatch the actual work.

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
58 {
59     NgbT ngb(tags);
60
61     if (description) {
62         HxString v(typename NgbT::IteratorCategory().toString());
63         v += ", ";
64         v += typename NgbT::PhaseCategory().toString();
65         description->setVariation(v);
66     }
67
68     HxFuncNgbOp2dExtraDispatch(dstPtr, srcPtr, extraPtr, dstSize, ngb);
69 }
```

The documentation for this class was generated from the following files:

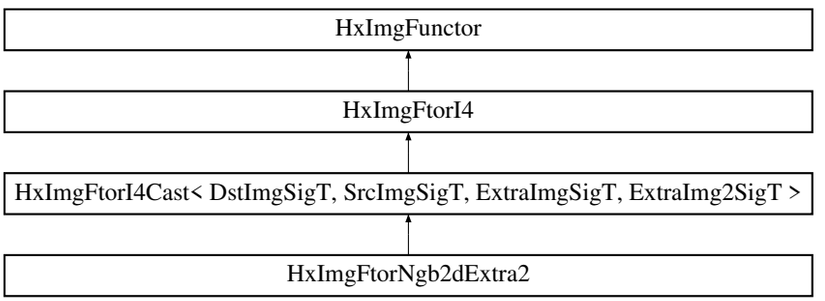
- **HxImgFtorNgb2dExtra.h**
- HxImgFtorNgb2dExtra.c

8.151 HxImgFtorNgb2dExtra2 Class Template Reference

Instantiation of generic algorithm for neighborhood operations on 2D images with two extra images.

`#include <HxImgFtorNgb2dExtra2.h>`

Inheritance diagram for HxImgFtorNgb2dExtra2::



Public Types

- typedef **HxImgFtorNgbExtra2Key KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorNgb2dExtra2** ()

  Constructor.

- virtual ∼**HxImgFtorNgb2dExtra2** ()

  Destructor.

- virtual bool **probeOp** (**HxTagList** &tags) const

  Probe for border size.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **Img3DataPtrType** extraPtr, **Img4DataPtrType** extra2Ptr, **HxSizes** dstSize, **HxSizes** srcSize, **HxSizes** extraSize, **HxSizes** extra2Size, **HxTagList** &tags, **HxImgFtorDescription** ∗description=0)

  Do it.

8.151.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class ExtraImg2SigT, class NgbT> class HxImgFtorNgb2dExtra2< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT, NgbT >

Instantiation of generic algorithm for neighborhood operations on 2D images with two extra images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- ExtraImgSigT is the signature type of the extra image
- ExtraImg2SigT is the signature type of the second extra image
- NgbT is the type of the neighbourhood functor

8.151.2 Member Typedef Documentation

8.151.2.1 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class ExtraImg2SigT, class NgbT> typedef HxImgFtorNgbExtra2Key HxImgFtorNgb2dExtra2::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI4Cast** (p. 773).

8.151.3 Constructor & Destructor Documentation

8.151.3.1 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class ExtraImg2SigT, class NgbT> HxImgFtorNgb2dExtra2< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT, NgbT >::HxImgFtorNgb2dExtra2 () [inline]

Constructor.

```
21     : HxImgFtorI4Cast<DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT>(
22         HxImgFtorNgbExtra2Key(
23             HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>(),
24             HxClassName<ExtraImgSigT>(), HxClassName<ExtraImg2SigT>(),
25             HxClassName<NgbT>()))
26 {
27     // Using ngb instead of something like ngbExtra2...
28     HxImgFtorRuleBase::instance().setResultType(
29         HxClassName<DstImgSigT>(), "ngb",
30         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
31     HxImgFtorRuleBase::instance().setExtraType(
32         HxClassName<ExtraImgSigT>(), "ngb",
33         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
34     HxImgFtorRuleBase::instance().setExtra2Type(
35         HxClassName<ExtraImg2SigT>(), "ngb",
36         HxClassName<SrcImgSigT>(), HxClassName<NgbT>());
37 }
```

8.151.3.2 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class ExtraImg2SigT, class NgbT> HxImgFtorNgb2dExtra2< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT, NgbT >::~HxImgFtorNgb2dExtra2 () [virtual]

Destructor.

```
43 {
44 }
```

8.151.4 Member Function Documentation

8.151.4.1 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class ExtraImg2SigT, class NgbT> bool HxImgFtorNgb2dExtra2< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT, NgbT >::probeOp (HxTagList & *tags*) const [virtual]

Probe for border size.

Reimplemented from **HxImgFunctor** (p. 864).

```
51 {
52     NgbT ngb(tags);
53     HxSizes borderSize = ngb.size() / HxSizes(2,2,2);
54     HxAddTag<HxSizes>(tags, "borderSize", borderSize);
55     return true;
56 }
```

8.151.4.2 template<class DstImgSigT, class SrcImgSigT, class ExtraImgSigT, class ExtraImg2SigT, class NgbT> void HxImgFtorNgb2dExtra2< DstImgSigT, SrcImgSigT, ExtraImgSigT, ExtraImg2SigT, NgbT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, Img3DataPtrType *extraPtr*, Img4DataPtrType *extra2Ptr*, HxSizes *dstSize*, HxSizes *srcSize*, HxSizes *extraSize*, HxSizes *extra2Size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

dstPtr Output image: IS = dstSize, IBS = 0

srcPtr Input image: IS = srcSize, IBS = taglist(borderSize)

extraPtr Extra image: IS = extraSize, IBS = taglist(borderSize)

extra2Ptr Extra image: IS = extra2Size, IBS = taglist(borderSize)

Calls **HxFuncNgbOp2dExtra2Dispatch** (p. 211) to dispatch the actual work.

Reimplemented from **HxImgFtorI4Cast** (p. 775).

```
67 {
68     NgbT ngb(tags);
69
70     if (description) {
71         HxString v(typename NgbT::IteratorCategory().toString());
72         v += ", ";
73         v += typename NgbT::PhaseCategory().toString();
74         description->setVariation(v);
75     }
76
77     HxFuncNgbOp2dExtra2Dispatch(dstPtr, srcPtr, extraPtr, extra2Ptr, dstSize, ngb);
78 }
```

The documentation for this class was generated from the following files:

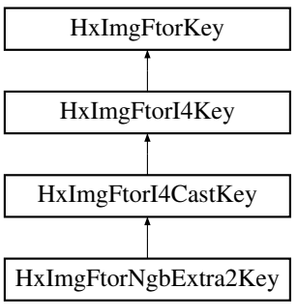
- **HxImgFtorNgb2dExtra2.h**
- HxImgFtorNgb2dExtra2.c

8.152 HxImgFtorNgbExtra2Key Class Reference

Key for HxImgFtorNgbExtra2.

`#include <HxImgFtorNgbExtra2Key.h>`

Inheritance diagram for HxImgFtorNgbExtra2Key::



Public Methods

- **HxImgFtorNgbExtra2Key** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** extraImgSig, **HxString** extraImg2Sig, **HxString** ngbName)

  Constructor.

8.152.1 Detailed Description

Key for HxImgFtorNgbExtra2.

8.152.2 Constructor & Destructor Documentation

8.152.2.1 HxImgFtorNgbExtra2Key::HxImgFtorNgbExtra2Key (HxString *dstImgSig*, HxString *srcImgSig*, HxString *extraImgSig*, HxString *extraImg2Sig*, HxString *ngbName*) [inline]

Constructor.

```
32     : HxImgFtorI4CastKey("HxImgFtorNgb2dExtra2", dstImgSig, srcImgSig,
33                          extraImgSig, extraImg2Sig)
34 {
35     addArgument(ngbName);
36 }
```

The documentation for this class was generated from the following file:

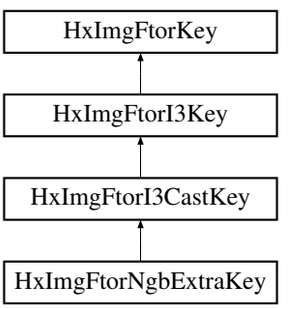
- **HxImgFtorNgbExtra2Key.h**

8.153 HxImgFtorNgbExtraKey Class Reference

Key for HxImgFtorNgbExtra.

`#include <HxImgFtorNgbExtraKey.h>`

Inheritance diagram for HxImgFtorNgbExtraKey::



Public Methods

- **HxImgFtorNgbExtraKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** extraImgSig, **HxString** ngbName)

Constructor.

8.153.1 Detailed Description

Key for HxImgFtorNgbExtra.

8.153.2 Constructor & Destructor Documentation

8.153.2.1 HxImgFtorNgbExtraKey::HxImgFtorNgbExtraKey (HxString *dstImgSig*, HxString *srcImgSig*, HxString *extraImgSig*, HxString *ngbName*) [inline]

Constructor.

```
30     : HxImgFtorI3CastKey("HxImgFtorNgb2dExtra", dstImgSig, srcImgSig, extraImgSig)
31 {
32     addArgument(ngbName);
33 }
```

The documentation for this class was generated from the following file:

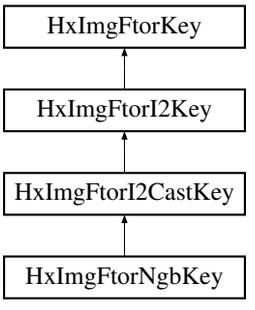
- **HxImgFtorNgbExtraKey.h**

8.154 HxImgFtorNgbKey Class Reference

Key for HxImgFtorNgb.

`#include <HxImgFtorNgbKey.h>`

Inheritance diagram for HxImgFtorNgbKey::



Public Methods

- **HxImgFtorNgbKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** ngbName)
  Constructor.

8.154.1 Detailed Description

Key for HxImgFtorNgb.

8.154.2 Constructor & Destructor Documentation

8.154.2.1 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**

8.155 HxImgFtorObserver Class Reference

Image functor observer.

`#include <HxImgFtorObserver.h>`

Public Methods

- virtual int **inserted** (const **HxImgFtorKey** &, const **HxImgFunctor** &)=0

  A <key,functor> combination has been inserted in **HxImgFtorTable** (p. 856).

8.155.1 Detailed Description

Image functor observer.

8.155.2 Member Function Documentation

8.155.2.1 virtual int HxImgFtorObserver::inserted (const HxImgFtorKey &, const HxImgFunctor &) [pure virtual]

A <key,functor> combination has been inserted in **HxImgFtorTable** (p. 856).

The documentation for this class was generated from the following file:

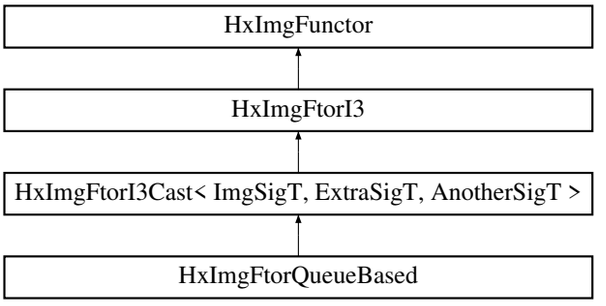
- **HxImgFtorObserver.h**

8.156 HxImgFtorQueueBased Class Template Reference

Instantiation of generic algorithm for queue based operations on 2d images.

`#include <HxImgFtorQueueBased.h>`

Inheritance diagram for HxImgFtorQueueBased::



Public Types

- typedef FunctorT::QT **QueueT**
- typedef FcvArray< PointT > **VecPointT**
- typedef FunctorT::VecNeighbors **VecNeighborT**

Public Methods

- **HxImgFtorQueueBased** ()
- ~**HxImgFtorQueueBased** ()
- virtual void **doIt** (**Img1DataPtrType** imgPtr, **Img2DataPtrType** extraPtr, **Img3DataPtrType** anotherPtr, **HxSizes** imgSize, **HxSizes** extraSize, **HxSizes** anotherSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  doIt is implemented by derived image functors:.

- void **fillNeighborValues** (const PointT &centralPoint, int connectivity, VecNeighborT &vecneighbors, const **HxSizes** &imgSize, const **Img1DataPtrType** &imgPtr, const **Img2DataPtrType** &extraPtr, const **Img3DataPtrType** &anotherPtr)

Static Public Methods

- **HxString className** ()
- void **fillNeighborValues** (PointT centralPoint, const VecPointT &neighborCoordinates, VecNeighborT &vecneighbors, **HxSizes** imgSize, **Img1DataPtrType** imgPtr, **Img2DataPtrType** extraPtr, **Img3DataPtrType** anotherPtr)
- void **createNeighborCoordinates** (VecPointT &neighborCoordinates, **HxTagList** &tags)

Public Attributes

- QueueT **queuet**

8.156.1 Detailed Description

template<class ImgSigT, class ExtraSigT, class AnotherSigT, class FunctorT> class HxImgFtorQueueBased< ImgSigT, ExtraSigT, AnotherSigT, FunctorT >

Instantiation of generic algorithm for queue based operations on 2d images.

8.156.2 Member Function Documentation

8.156.2.1 template<class ImgSigT, class ExtraSigT, class AnotherSigT, class FunctorT> void HxImgFtorQueueBased< ImgSigT, ExtraSigT, AnotherSigT, FunctorT >::doIt (Img1DataPtrType *imgPtr*, Img2DataPtrType *extraPtr*, Img3DataPtrType *anotherPtr*, HxSizes *imgSize*, HxSizes *extraSize*, HxSizes *anotherSize*, HxTagList & *tags*, HxImgFtorDescription * *f* = 0) [virtual]

doIt is implemented by derived image functors:.

- **HxImgFtorBpo::doIt** (p. 703)
- **HxImgFtorGenConv2d::doIt** (p. 713)
- **HxImgFtorGenConv2dK1d::doIt** (p. 716)
- **HxImgFtorGenConv3d::doIt** (p. 723)
- **HxImgFtorGenConv3dK1d::doIt** (p. 725)
- **HxImgFtorKernelNgb2d::doIt** (p. 797)

Reimplemented from **HxImgFtorI3Cast** (p. 764).

```
141                                 {
142
143     SHOW(imgSize); SHOW(extraSize); SHOW(anotherSize); SHOW(tags);
144     if (!(imgSize==extraSize && imgSize==anotherSize)) {
145         HxString err("ERROR: unequal sizes in ");
146 //      err+=typeid(*this).name();
147         throw err;
148     }
149     HxAddTag(tags, "imgSize", imgSize); // need it for debugging
150
151
152     int imgsizex = imgSize.x();
153     int imgsizey = imgSize.y();
154     int imgsizez = imgSize.z(); if (imgsizez!=1) throw HxString("QueueBeased pattern not (yet) certified for imgsize
155     FunctorT functor(tags, imgsizex, imgsizey);
156     SHOW(typeid(*this).name());
157
158     int connectivity=HxGetTag(tags, "connectivity", 8);
159 //  std::cout << "connectivity=" <<connectivity<<std::endl;
160
161     // GLOBAL INIT
162     { // local scope
163         if (!queuet.empty()) { // debug only
164             SHOW(queuet.size()); // uhhhhhhhhhhhhhhhhhhh
165             queuet.clear();
166         }
167         Img1DataPtrType d=imgPtr;
168         Img2DataPtrType s1=extraPtr;
169         Img3DataPtrType s2=anotherPtr;
170         for (int y=0; y<imgsizey; y++) {
171             for (int x=0 ; x<imgsizex; x++) {
172                 typename FunctorT::PointValueT vp;
173                 PointT p(x, y);
174                 vp.point=p;
175                 typename ImgSigT::ArithType dest=d.read();
176                 typename ExtraSigT::ArithType src1=s1.read();
177                 typename AnotherSigT::ArithType src2=s2.read();
178                 bool q=functor.globalPixelInit(vp, dest, src1, src2);
179                 if (q) queuet.push(vp); // NOTE: bool ok=queue.insert(vp).second is NEVER false
180                 d.write(dest);
181                 d.incX(); s1.incX(); s2.incX();
```

```
182             }
183             d.decX(imgsizex); s1.decX(imgsizex); s2.decX(imgsizex);
184             d.incY(); s1.incY(); s2.incY();
185         }
186     } // end local scope
187
188     // NEW: array of points to simplify loop 4/8/... connectivity, now skip (0,0,0)
189     FcvArray<PointT> neighborCoordinates;
190     createNeighborCoordinates(neighborCoordinates, tags);
191
192     // clear speedup startpoint for localPixelInit() in large do while loop
193     //__asm int 3
194     PointT startpoint(0,0);
195
196     typename FunctorT::VecNeighbors vecneighbors;
197
198
199     // LOOP until done with INIT local + PROPAGATE
200     do {
201
202         // INIT: FILL QUEUE
203         { // local scope
204             bool freshstart=functor.wantFreshStartLocalPixelInit();
205             if (freshstart) {
206                 startpoint.x=0;
207                 startpoint.y=0;
208             }
209             Img1DataPtrType d=imgPtr;
210             Img2DataPtrType s1=extraPtr;
211             Img3DataPtrType s2=anotherPtr;
212             bool continueloop=true;
213             int x=startpoint.x;
214             int y=startpoint.y;
215             d.incXYZ(x, y);
216             s1.incXYZ(x, y);
217             s2.incXYZ(x, y);
218             bool wenttonewpoint=false;
219             if (wenttonewpoint) y=0;
220             for (; y<imgsizey; y++) {
221                 if (wenttonewpoint) x=0;
222                 for (; x<imgsizex; x++) {
223                     if (!wenttonewpoint) {
224                         wenttonewpoint=true;
225                     }
226                     typename FunctorT::PointValueT vp;
227                     vp.point.x=x;
228                     vp.point.y=y;
229                     typename ImgSigT::ArithType dest=d.read();
230                     typename ExtraSigT::ArithType src1=s1.read();
231                     typename AnotherSigT::ArithType src2=s2.read();
232                     bool q=functor.localPixelInit(vp, dest, src1, src2, continueloop);
233                     if (q) queuet.push(vp); // NOTE: bool ok=queue.insert(vp).second is NEVER false
234                     d.write(dest);
235                     if (!continueloop) break;
236                     d.incX(); s1.incX(); s2.incX();
237                 }
238                 if (!continueloop) break;
239                 d.decX(imgsizex); s1.decX(imgsizex); s2.decX(imgsizex);
240                 d.incY(); s1.incY(); s2.incY();
241             }
242             // rememmber startpoint for next time
243             startpoint.x=x;
244             startpoint.y=y;
245
246         } // end local scope
```

```
247
248         // PROPAGATION
249         { // local scope
250             while (!queuet.empty()) { // line 4
251                 typename FunctorT::PointValueT vp(queuet.top()); queuet.pop(); // line 5
252
253                 // give functor first=this=center data
254                 { // local scope
255                     int tempx=vp.point.x, tempy=vp.point.y;
256                     Img1DataPtrType d=imgPtr;
257                     Img2DataPtrType s1=extraPtr;
258                     Img3DataPtrType s2=anotherPtr;
259                     d.incXYZ(tempx, tempy);
260                     s1.incXYZ(tempx, tempy);
261                     s2.incXYZ(tempx, tempy);
262                     functor.first(vp, d.read(), s1.read(), s2.read());
263                 } // local scope
264
265                 // give functor the neighbour pixels
266                 // NOT yet optimized for iterator++ or LUT for coords
267                 // NEW: use vector to pass neighbors, here be compatible with sequential code
268                 { //local scope
269                     fillNeighborValues(
270                         vp.point,
271                         neighborCoordinates,
272                         //connectivity,
273                         vecneighbors,
274                         imgSize,
275                         imgPtr,
276                         extraPtr,
277                         anotherPtr);
278                     // CALCULATE put functor to work, might be removed by first call to getdata()
279                     functor.calculate(vecneighbors);
280                 } //local scope
281
282
283
284                 // read out phase, might have many values of different types
285                 { // local scope
286                     typename FunctorT::ActionT action;
287                     while ((action=functor.result3())!=FunctorT::stopAction) {
288                         switch (action) {
289                         case FunctorT::queueAction:
290                             { // local scope
291                                 typename FunctorT::PointValueT vp;
292                                 functor.getItemToQueue(vp);
293                                 queuet.push(vp);
294                             } // local scope
295                             break;
296                         case FunctorT::removeAction:
297                             { // local scope
298                                 typename FunctorT::PointValueT low, high;
299                                 functor.getItemToRemove(low, high);
300                                 typename QueueT::iterator lowiter=queuet.lower_bound(low);
301                                 typename QueueT::iterator highiter=queuet.after_upper_bound(high);
302                                 for (typename QueueT::iterator iter=lowiter; iter!=highiter; iter++) {
303                                     bool k=functor.killThisOne(*iter);
304                                     if (k) {
305                                         queuet.erase(iter);
306                                         break;
307                                     }
308                                 }
309                             } // local scope
310                             break;
311                         case FunctorT::writeAction:
```

```
312                             { // local scope
313                                 Img1DataPtrType p=imgPtr;
314                                 PointT point;
315                                 typename ImgSigT::ArithType arith;
316                                 functor.getItemToWrite(point, arith);
317                                 p.incXYZ(point.x, point.y);
318                                 p.write(arith);
319                             } // local scope
320                             break;
321                         default: throw HxString("ERROR switch (action) default:");
322                         }; // switch (action)
323                     } // while ((action=functor.result3())!=FunctorT::stopAction)
324                 } // local scope
325             } // while (!queuet.empty())
326         } // end local scope PROPAGATION
327     } while (functor.wantAnotherLoop());
328
329
330 }
```

The documentation for this class was generated from the following files:

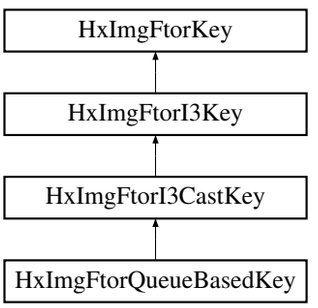
- **HxImgFtorQueueBased.h**
- HxImgFtorQueueBased.c

8.157 HxImgFtorQueueBasedKey Class Reference

Key for **HxImgFtorQueueBased** (p. 822).

`#include <HxImgFtorQueueBasedKey.h>`

Inheritance diagram for HxImgFtorQueueBasedKey::



Public Methods

- **HxImgFtorQueueBasedKey** (**HxString** imgSig, **HxString** extraSig, **HxString** kerImgSig, **HxString** FunctorName)

8.157.1 Detailed Description

Key for **HxImgFtorQueueBased** (p. 822).

The documentation for this class was generated from the following file:

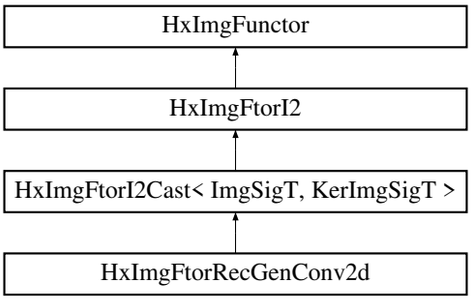
- **HxImgFtorQueueBasedKey.h**

8.158 HxImgFtorRecGenConv2d Class Template Reference

Instantiation of generic algorithm for recursive generalized convolution on 2D images.

`#include <HxImgFtorRecGenConv2d.h>`

Inheritance diagram for HxImgFtorRecGenConv2d::



Public Types

- typedef **HxImgFtorRecGenConvKey KeyType**
  The key type of this class.

Public Methods

- **HxImgFtorRecGenConv2d** ()
  Constructor.
- virtual ~**HxImgFtorRecGenConv2d** ()
  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** imgPtr, **Img2DataPtrType** kerPtr, **HxSizes** imgSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)
  Do it.

8.158.1 Detailed Description

template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> class HxImgFtorRecGenConv2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >

Instantiation of generic algorithm for recursive generalized convolution on 2D images.

Template parameters:

- ImgSigT is the signature type of the source/destination image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor

8.158.2 Member Typedef Documentation

8.158.2.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> typedef HxImgFtorRecGenConvKey HxImgFtorRecGenConv2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.158.3 Constructor & Destructor Documentation

8.158.3.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> HxImgFtorRecGenConv2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::HxImgFtorRecGenConv2d ()

Constructor.

```
30         : HxImgFtorI2Cast<ImgSigT, KerImgSigT>(
31             HxImgFtorRecGenConvKey(
32                 HxClassName<ImgSigT>(), HxClassName<KerImgSigT>(),
33                 HxClassName<PixOpT>(), HxClassName<RedOpT>()))
34 {
35 #ifdef CD_TRACE
36     HxEnvironment::instance()->outputStream()
37         << "HxImgFtorRecGenConv2d::HxImgFtorRecGenConv2d()" << STD_ENDL;
38 #endif
39 }
```

8.158.3.2 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> HxImgFtorRecGenConv2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::~HxImgFtorRecGenConv2d () [virtual]

Destructor.

```
44 {
45 #ifdef CD_TRACE
46     HxEnvironment::instance()->outputStream()
47         << "HxImgFtorRecGenConv2d::~HxImgFtorRecGenConv2d()" << STD_ENDL;
48 #endif
49 }
```

8.158.4 Member Function Documentation

8.158.4.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> void HxImgFtorRecGenConv2d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::doIt (Img1DataPtrType *imgPtr*, Img2DataPtrType *kerPtr*, HxSizes *imgSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

imgPtr Input/Output image: IS = imgSize, IBS = kerSize/2

kerPtr Input image, IS = kerSize, IBS = 0

Calls HxFuncRecGenConvOp2dDispatch to dispatch the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
58 {
59     PixOpT pixOp(tags);
60     RedOpT redOp(tags);
61
62     typedef typename KerImgSigT::ArithType ArithType;
63     ArithType dummy;
64
65     HxFuncRecGenConv2dDispatch(imgPtr, kerPtr, dummy, imgSize,
66                                kerSize, pixOp, redOp);
67 }
```

The documentation for this class was generated from the following files:

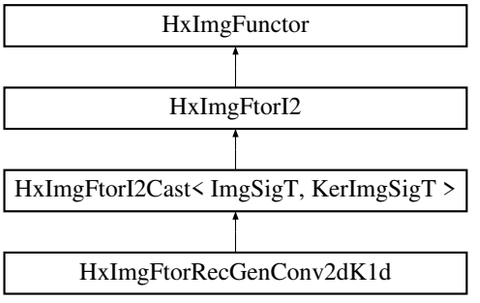
- **HxImgFtorRecGenConv2d.h**
- HxImgFtorRecGenConv2d.c

8.159 HxImgFtorRecGenConv2dK1d Class Template Reference

Instantiation of generic algorithm for recursive generalized convolution on 2D images with a 1d kernel.

`#include <HxImgFtorRecGenConv2dK1d.h>`

Inheritance diagram for HxImgFtorRecGenConv2dK1d::



Public Types

- typedef **HxImgFtorRecGenConvK1dKey KeyType**

  The key type of this class.

Public Methods

- **HxImgFtorRecGenConv2dK1d** ()

  Constructor.

- virtual ∼**HxImgFtorRecGenConv2dK1d** ()

  Destructor.

Protected Methods

- virtual void **doIt** (**Img1DataPtrType** imgPtr, **Img2DataPtrType** kerPtr, **HxSizes** imgSize, **HxSizes** kerSize, **HxTagList** &tags, **HxImgFtorDescription** ∗description=0)

  Do it.

8.159.1 Detailed Description

template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> class HxImgFtorRecGenConv2dK1d< ImgSigT, KerImgSigT, PixOpT, RedOpT >

Instantiation of generic algorithm for recursive generalized convolution on 2D images with a 1d kernel.

Template parameters:

- ImgSigT is the signature type of the source/destination image
- KerImgSigT is the signature type of the kernel image
- PixOpT is the type of the pixel combining functor
- RedOpT is the type of the pixel reducing functor

8.159.2 Member Typedef Documentation

8.159.2.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> typedef HxImgFtorRecGenConvK1dKey HxImgFtorRecGenConv2dK1d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.159.3 Constructor & Destructor Documentation

8.159.3.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> HxImgFtorRecGenConv2dK1d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::HxImgFtorRecGenConv2dK1d ()

Constructor.

```
30         : HxImgFtorI2Cast<ImgSigT, KerImgSigT>(
31             HxImgFtorRecGenConvK1dKey(
32                 HxClassName<ImgSigT>(), HxClassName<KerImgSigT>(),
33                 HxClassName<PixOpT>(), HxClassName<RedOpT>()))
34 {
35 #ifdef CD_TRACE
36     HxEnvironment::instance()->outputStream()
37         << "HxImgFtorRecGenConv2dK1d::HxImgFtorRecGenConv2dK1d()" << STD_ENDL;
38 #endif
39 }
```

8.159.3.2 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> HxImgFtorRecGenConv2dK1d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::~HxImgFtorRecGenConv2dK1d () [virtual]

Destructor.

```
45 {
46 #ifdef CD_TRACE
47     HxEnvironment::instance()->outputStream()
48         << "HxImgFtorRecGenConv2dK1d::~HxImgFtorRecGenConv2dK1d()" << STD_ENDL;
49 #endif
50 }
```

8.159.4 Member Function Documentation

8.159.4.1 template<class ImgSigT, class KerImgSigT, class PixOpT, class RedOpT> void HxImgFtorRecGenConv2dK1d< ImgSigT, KerImgSigT, PixOpT, RedOpT >::doIt (Img1DataPtrType *imgPtr*, Img2DataPtrType *kerPtr*, HxSizes *imgSize*, HxSizes *kerSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

imgPtr Input/Output image: IS = imgSize, IBS = taglist(borderSize)

kerPtr Input image, IS = kerSize, IBS = 0

Calls **HxFuncRecGenConv2dK1dDispatch** (p. 219) to dispatch the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
59 {
60     int dimension = HxGetTag(tags, "dimension", 0);
61     bool buffered = HxGetTag(tags, "buffered", false);
62
```

```
63     if (buffered && description)
64         description->setVariation("buffered");
65
66     HxSizes borderSize = HxGetTag(tags, "borderSize", HxSizes(0,0,0));
67
68     PixOpT pixOp(tags);
69     RedOpT redOp(tags);
70
71     typedef typename KerImgSigT::ArithType ArithType;
72     ArithType dummy;
73
74     HxFuncRecGenConv2dK1dDispatch(imgPtr, kerPtr, dummy, imgSize, borderSize,
75                                   kerSize, pixOp, redOp, dimension, buffered);
76
77 }
```

The documentation for this class was generated from the following files:

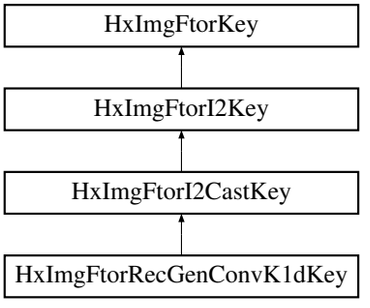
- **HxImgFtorRecGenConv2dK1d.h**
- HxImgFtorRecGenConv2dK1d.c

8.160 HxImgFtorRecGenConvK1dKey Class Reference

Key for HxImgFtorRecGenConvK1d.

`#include <HxImgFtorRecGenConvK1dKey.h>`

Inheritance diagram for HxImgFtorRecGenConvK1dKey::



Public Methods

- **HxImgFtorRecGenConvK1dKey** (**HxString** imgSig, **HxString** kerImgSig, **HxString** pixOpName, **HxString** redOpName)

  Constructor.

8.160.1 Detailed Description

Key for HxImgFtorRecGenConvK1d.

8.160.2 Constructor & Destructor Documentation

8.160.2.1 HxImgFtorRecGenConvK1dKey::HxImgFtorRecGenConvK1dKey (HxString *imgSig*, HxString *kerImgSig*, HxString *pixOpName*, HxString *redOpName*) [inline]

Constructor.

```
32         : HxImgFtorI2CastKey("HxImgFtorRecGenConvK1d", imgSig, kerImgSig)
33 {
34     addArgument(pixOpName);
35     addArgument(redOpName);
36 }
```

The documentation for this class was generated from the following file:

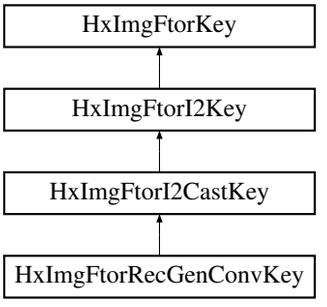
- **HxImgFtorRecGenConvK1dKey.h**

8.161 HxImgFtorRecGenConvKey Class Reference

Key for HxImgFtorRecGenConv.

`#include <HxImgFtorRecGenConvKey.h>`

Inheritance diagram for HxImgFtorRecGenConvKey::



Public Methods

- **HxImgFtorRecGenConvKey** (**HxString** imgSig, **HxString** kerImgSig, **HxString** pixOpName, **HxString** redOpName)

  Constructor.

8.161.1 Detailed Description

Key for HxImgFtorRecGenConv.

8.161.2 Constructor & Destructor Documentation

8.161.2.1 HxImgFtorRecGenConvKey::HxImgFtorRecGenConvKey (HxString *imgSig*, HxString *kerImgSig*, HxString *pixOpName*, HxString *redOpName*) [inline]

Constructor.

```
32         : HxImgFtorI2CastKey("HxImgFtorRecGenConv", imgSig, kerImgSig)
33 {
34     addArgument(pixOpName);
35     addArgument(redOpName);
36 }
```

The documentation for this class was generated from the following file:

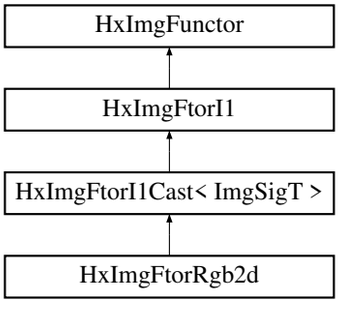
- **HxImgFtorRecGenConvKey.h**

8.162 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)

  Do it.

8.162.1 Detailed Description

template<class ImgSigT, class RgbT> class HxImgFtorRgb2d< ImgSigT, RgbT >

Instantiation of generic algorithm for display of 2d images.

Template parameters:

- ImgSigT is the signature type of the image
- RgbT is the type of the rgb pixel functor

8.162.2 Member Typedef Documentation

8.162.2.1 template<class ImgSigT, class RgbT> typedef HxImgFtorRgbKey HxImgFtorRgb2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 732).

8.162.3 Constructor & Destructor Documentation

8.162.3.1 template<class ImgSigT, class RgbT> HxImgFtorRgb2d< ImgSigT, RgbT >::HxImgFtorRgb2d () [inline]

Constructor.

```
20     : HxImgFtorI1Cast<ImgSigT>(HxImgFtorRgbKey(HxClassName<ImgSigT>(),
21                                HxClassName<RgbT>()))
22 {
23 #ifdef CD_TRACE
24     HxEnvironment::instance()->outputStream()
25         << "HxImgFtorRgb2d::HxImgFtorRgb2d()" << STD_ENDL;
26 #endif
27     static HxRegKey* surKey
28         = HxRegistry::instance().insertKey("/imagefunctortable/rgb");
29     surKey->insertValue(HxClassName<RgbT>(), HxRegData(1));
30 }
```

8.162.3.2 template<class ImgSigT, class RgbT> HxImgFtorRgb2d< ImgSigT, RgbT >::~HxImgFtorRgb2d () [virtual]

Destructor.

```
34 {
35 #ifdef CD_TRACE
36     HxEnvironment::instance()->outputStream()
37         << "HxImgFtorRgb2d::~HxImgFtorRgb2d()" << STD_ENDL;
38 #endif
39 }
```

8.162.4 Member Function Documentation

8.162.4.1 template<class ImgSigT, class RgbT> void HxImgFtorRgb2d< ImgSigT, RgbT >::doIt (ImgDataPtrType *ptr*, HxSizes *size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

ptr Input image: IS = size, IBS = 0

Calls **HxFuncRgbOp2d** (p. 220) to do the actual work.

Reimplemented from **HxImgFtorI1Cast** (p. 734).

```
45 {
46     int* pixels = HxGetTag<int*>(tags, "pixels");
47     int resWidth = HxGetTag<int>(tags, "resWidth");
48     int resHeight = HxGetTag<int>(tags, "resHeight");
49     HxGeoIntType gi = HxGetTag<HxGeoIntType>(tags, "gi");
50
51     RgbT rgb(tags);
52
53     HxFuncRgbOp2d(ptr, size, pixels, resWidth, resHeight, gi, rgb);
54 }
```

The documentation for this class was generated from the following files:

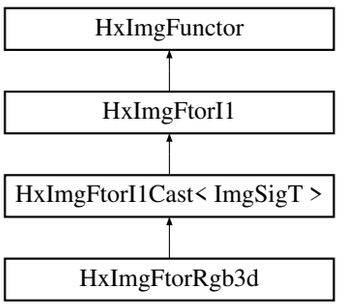
- **HxImgFtorRgb2d.h**
- HxImgFtorRgb2d.c

8.163 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)
  Do it.

8.163.1 Detailed Description

template<class ImgSigT, class RgbT> class HxImgFtorRgb3d< ImgSigT, RgbT >

Instantiation of generic algorithm for display of 3d images.

Template parameters:

- ImgSigT is the signature type of the image
- RgbT is the type of the rgb pixel functor

8.163.2 Member Typedef Documentation

8.163.2.1 template<class ImgSigT, class RgbT> typedef HxImgFtorRgbKey HxImgFtorRgb3d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 732).

8.163.3 Constructor & Destructor Documentation

8.163.3.1 template<class ImgSigT, class RgbT> HxImgFtorRgb3d< ImgSigT, RgbT >::HxImgFtorRgb3d () [inline]

Constructor.

```
20     : HxImgFtorI1Cast<ImgSigT>(HxImgFtorRgbKey(HxClassName<ImgSigT>(),
21                                HxClassName<RgbT>()))
22 {
23 #ifdef CD_TRACE
24     HxEnvironment::instance()->outputStream()
25         << "HxImgFtorRgb3d::HxImgFtorRgb3d()" << STD_ENDL;
26 #endif
27     static HxRegKey* surKey
28         = HxRegistry::instance().insertKey("/imagefunctortable/rgb");
29     surKey->insertValue(HxClassName<RgbT>(), HxRegData(1));
30 }
```

8.163.3.2 template<class ImgSigT, class RgbT> HxImgFtorRgb3d< ImgSigT, RgbT >::~HxImgFtorRgb3d () [virtual]

Destructor.

```
34 {
35 #ifdef CD_TRACE
36     HxEnvironment::instance()->outputStream()
37         << "HxImgFtorRgb3d::~HxImgFtorRgb3d()" << STD_ENDL;
38 #endif
39 }
```

8.163.4 Member Function Documentation

8.163.4.1 template<class ImgSigT, class RgbT> void HxImgFtorRgb3d< ImgSigT, RgbT >::doIt (ImgDataPtrType *ptr*, HxSizes *size*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Do it.

Parameters:

ptr Input image: IS = size, IBS = 0

Calls **HxFuncRgbOp3d** (p. 221) to do the actual work.

Reimplemented from **HxImgFtorI1Cast** (p. 734).

```
45 {
46     int* pixels = HxGetTag<int*>(tags, "pixels");
47     int dimension = HxGetTag<int>(tags, "dimension");
48     int coordinate = HxGetTag<int>(tags, "coordinate");
49     int resWidth = HxGetTag<int>(tags, "resWidth");
50     int resHeight = HxGetTag<int>(tags, "resHeight");
51     HxGeoIntType gi = HxGetTag<HxGeoIntType>(tags, "gi");
52
53     RgbT rgb(tags);
54
55     HxFuncRgbOp3d(ptr, size, pixels, dimension, coordinate, resWidth, resHeight,
56                   gi, rgb);
57 }
```

The documentation for this class was generated from the following files:

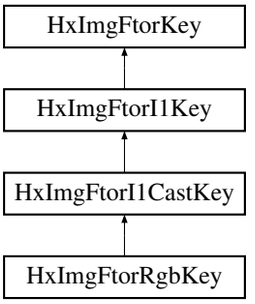
- **HxImgFtorRgb3d.h**
- HxImgFtorRgb3d.c

8.164 HxImgFtorRgbKey Class Reference

Key for HxImgFtorRgb.

`#include <HxImgFtorRgbKey.h>`

Inheritance diagram for HxImgFtorRgbKey::



Public Methods

- **HxImgFtorRgbKey** (**HxString** imgSig, **HxString** rgbName)
  Constructor.

8.164.1 Detailed Description

Key for HxImgFtorRgb.

8.164.2 Constructor & Destructor Documentation

8.164.2.1 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**

8.165 HxImgFtorRuleBase Class Reference

Rule base for storage and retrieval of existing image types in instantiated image functors.

`#include <HxImgFtorRuleBase.h>`

Public Types

- typedef **HxIfRbPair QueryResultType**

  The type of the result of a query.

Public Methods

- void **setResultType** (**HxString** resultType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Insert ftorClass<srcType,opName>=resultType under [/imagefunctor/rulebase/resulttype].

- void **setResultType** (**HxString** resultType, **HxString** ftorClass, **HxString** src1Type, **HxString** src2Type, **HxString** opName)

  Insert ftorClass<src1Type,src2Type,opName>=resultType under [/imagefunctor/rulebase/resulttype].

- **QueryResultType getResultType** (**HxImageSignature** defaultResult, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/resulttype].

- **QueryResultType getResultType** (**HxImageSignature** defaultResult, **HxString** ftorClass, **HxString** src1Type, **HxString** src2Type, **HxString** opName)

  Get value for ftorClass<src1Type,src2Type,opName> from [/imagefunctor/rulebase/resulttype].

- void **setArgumentType** (**HxString** argumentType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Insert ftorClass<srcType,opName>=argumentType under [/imagefunctor/rulebase/argumenttype].

- **QueryResultType getArgumentType** (**HxImageSignature** defaultArgument, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/argumenttype].

- void **setKernelType** (**HxString** kernelType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Insert ftorClass<srcType,opName>=kernelType under [/imagefunctor/rulebase/kerneltype].

- **QueryResultType getKernelType** (**HxImageSignature** defaultKernel, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/kerneltype].

- void **setExtraType** (**HxString** extraType, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Insert ftorClass<srcType,opName>=extraType under [/imagefunctor/rulebase/extratype].

- **QueryResultType getExtraType** (**HxImageSignature** defaultExtra, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/extratype].

- void **setExtra2Type** (**HxString** extra2Type, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

Insert ftorClass<srcType,opName>=extra2Type under [/imagefunctor/rulebase/extra2type].

- **QueryResultType getExtra2Type** (**HxImageSignature** defaultExtra2, **HxString** ftorClass, **HxString** srcType, **HxString** opName)

  Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/extra2type].

- void **setIsModifying** (**HxString** ftorClass, **HxString** opName, bool f=true)

  Insert ftorClass<opName>=f under [/imagefunctor/rulebase/isModifying].

- bool **getIsModifying** (**HxString** ftorClass, **HxString** opName)

  Get value for ftorClass<opName> from [/imagefunctor/rulebase/isModifying].

- ~**HxImgFtorRuleBase** ()

  Destructor.

Static Public Methods

- HxImgFtorRuleBase & **instance** ()

  Access to this singleton class.

8.165.1 Detailed Description

Rule base for storage and retrieval of existing image types in instantiated image functors.

Rules are stored in the registry under key [”/imagefunctor/rulebase”]. Current list:

efunctor/rulebase/resulttype
nctor/rulebase/argumenttype
efunctor/rulebase/kerneltype
gefunctor/rulebase/extratype
unctor/rulebase/isModifying

8.165.2 Member Typedef Documentation

8.165.2.1 typedef HxIfRbPair HxImgFtorRuleBase::QueryResultType

The type of the result of a query.

8.165.3 Constructor & Destructor Documentation

8.165.3.1 HxImgFtorRuleBase::~HxImgFtorRuleBase ()

Destructor.

```
24 {
25 }
```

8.165.4 Member Function Documentation

8.165.4.1 HxImgFtorRuleBase & HxImgFtorRuleBase::instance () [static]

Access to this singleton class.

```
29 {
30     static HxImgFtorRuleBase theRuleBase;
31     return theRuleBase;
32 }
```

8.165.4.2 void HxImgFtorRuleBase::setResultType (HxString *resultType*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Insert ftorClass<srcType,opName>=resultType under [/imagefunctor/rulebase/resulttype].

```
80 {
81     HxStringList argList;
82     argList << srcType << opName;
83     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
84     setRule("resulttype", key, resultType);
85 }
```

8.165.4.3 void HxImgFtorRuleBase::setResultType (HxString *resultType*, HxString *ftorClass*, HxString *src1Type*, HxString *src2Type*, HxString *opName*)

Insert ftorClass<src1Type,src2Type,opName>=resultType under [/imagefunctor/rulebase/resulttype].

```
91 {
92     HxStringList argList;
93     argList << src1Type << src2Type << opName;
94     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
95     setRule("resulttype", key, resultType);
96 }
```

8.165.4.4 HxIfRbPair HxImgFtorRuleBase::getResultType (HxImageSignature *defaultResult*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/resulttype].

Return defaultResult if not found.

```
102 {
103     HxStringList argList;
104     argList << srcType << opName;
105     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
106
107     HxString result = getRule("resulttype", key);
108     if (result != HxString(""))
109         return HxImageSignature::NameToSignature(result);
110     else
111         return HxIfRbPair(defaultResult, false);
112 }
```

8.165.4.5 HxIfRbPair HxImgFtorRuleBase::getResultType (HxImageSignature *defaultResult*, HxString *ftorClass*, HxString *src1Type*, HxString *src2Type*, HxString *opName*)

Get value for ftorClass<src1Type,src2Type,opName> from [/imagefunctor/rulebase/resulttype].

Return defaultResult if not found.

```
118 {
119     HxStringList argList;
120     argList << src1Type << src2Type << opName;
121     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
122 
123     HxString result = getRule("resulttype", key);
124     if (result != HxString(""))
125         return HxImageSignature::NameToSignature(result);
126     else
127         return HxIfRbPair(defaultResult, false);
128 }
```

8.165.4.6 void HxImgFtorRuleBase::setArgumentType (HxString *argumentType*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Insert ftorClass<srcType,opName>=argumentType under [/imagefunctor/rulebase/argumenttype].

```
134 {
135     HxStringList argList;
136     argList << srcType << opName;
137     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
138     setRule("argumenttype", key, argumentType);
139 }
```

8.165.4.7 HxIfRbPair HxImgFtorRuleBase::getArgumentType (HxImageSignature *defaultArgument*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/argumenttype].

Return defaultResult if not found.

```
145 {
146     HxStringList argList;
147     argList << srcType << opName;
148     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
149 
150     HxString argument = getRule("argumenttype", key);
151     if (argument != HxString(""))
152         return HxImageSignature::NameToSignature(argument);
153     else
154         return HxIfRbPair(defaultArgument, false);
155 }
```

8.165.4.8 void HxImgFtorRuleBase::setKernelType (HxString *kernelType*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Insert ftorClass<srcType,opName>=kernelType under [/imagefunctor/rulebase/kerneltype].

```
161 {
162     HxStringList argList;
163     argList << srcType << opName;
164     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
165     setRule("kerneltype", key, kernelType);
166 }
```

8.165.4.9 HxIfRbPair HxImgFtorRuleBase::getKernelType (HxImageSignature *defaultKernel*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/kerneltype].

Return defaultResult if not found.

```
172 {
173     HxStringList argList;
174     argList << srcType << opName;
175     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
176 
177     HxString result = getRule("kerneltype", key);
178     if (result != HxString(""))
179         return HxImageSignature::NameToSignature(result);
180     else
181         return HxIfRbPair(defaultKernel, false);
182 }
```

8.165.4.10 void HxImgFtorRuleBase::setExtraType (HxString *extraType*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Insert ftorClass<srcType,opName>=extraType under [/imagefunctor/rulebase/extratype].

```
188 {
189     HxStringList argList;
190     argList << srcType << opName;
191     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
192     setRule("extratype", key, extraType);
193 }
```

8.165.4.11 HxIfRbPair HxImgFtorRuleBase::getExtraType (HxImageSignature *defaultExtra*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/extratype].

Return defaultResult if not found.

```
199 {
200     HxStringList argList;
201     argList << srcType << opName;
202     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
203 
204     HxString result = getRule("extratype", key);
205     if (result != HxString(""))
206         return HxImageSignature::NameToSignature(result);
207     else
208         return HxIfRbPair(defaultExtra, false);
209 }
```

8.165.4.12 void HxImgFtorRuleBase::setExtra2Type (HxString *extra2Type*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Insert ftorClass<srcType,opName>=extra2Type under [/imagefunctor/rulebase/extra2type].

```
215 {
216     HxStringList argList;
217     argList << srcType << opName;
218     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
219     setRule("extra2type", key, extra2Type);
220 }
```

8.165.4.13 HxIfRbPair HxImgFtorRuleBase::getExtra2Type (HxImageSignature *defaultExtra2*, HxString *ftorClass*, HxString *srcType*, HxString *opName*)

Get value for ftorClass<srcType,opName> from [/imagefunctor/rulebase/extra2type].

Return defaultResult if not found.

```
226 {
227     HxStringList argList;
228     argList << srcType << opName;
229     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
230 
231     HxString result = getRule("extra2type", key);
232     if (result != HxString(""))
233         return HxImageSignature::NameToSignature(result);
234     else
235         return HxIfRbPair(defaultExtra2, false);
236 }
```

8.165.4.14 void HxImgFtorRuleBase::setIsModifying (HxString *ftorClass*, HxString *opName*, bool *f* = true)

Insert ftorClass<opName>=f under [/imagefunctor/rulebase/isModifying].

```
241 {
242     HxStringList argList;
243     argList << opName;
244     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
245     setFlagRule("isModifying", key, f);
246 }
```

8.165.4.15 bool HxImgFtorRuleBase::getIsModifying (HxString *ftorClass*, HxString *opName*)

Get value for ftorClass<opName> from [/imagefunctor/rulebase/isModifying].

```
251 {
252     HxStringList argList;
253     argList << opName;
254     HxImgFtorKey key(ftorClass, argList.begin(), argList.end());
255     return getFlagRule("isModifying", key);
256 }
```

The documentation for this class was generated from the following files:

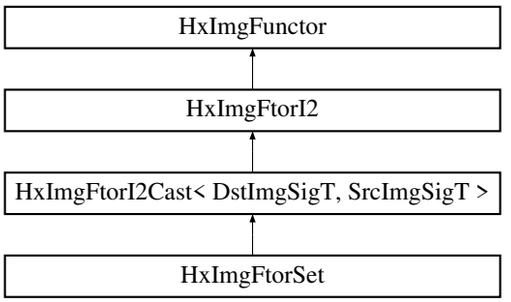
- **HxImgFtorRuleBase.h**
- HxImgFtorRuleBase.c

8.166 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** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **HxSizes** srcSize, **HxSizes** dstSize, **HxTagList** &tags, **HxImgFtorDescription** *=0)

  Calls **HxFuncSet** (p. 223) *to do the actual work.*

8.166.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT> class HxImgFtorSet< DstImgSigT, SrcImgSigT >

Instantiation of generic algorithm for set operations on images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image

8.166.2 Member Typedef Documentation

8.166.2.1 template<class DstImgSigT, class SrcImgSigT> typedef HxImgFtorSetKey HxImgFtorSet::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.166.3 Constructor & Destructor Documentation

8.166.3.1 template<class DstImgSigT, class SrcImgSigT> HxImgFtorSet< DstImgSigT, SrcImgSigT >::HxImgFtorSet () [inline]

Constructor.

```
22     : HxImgFtorI2Cast<DstImgSigT, SrcImgSigT>(
23         HxImgFtorSetKey(HxClassName<DstImgSigT>(), HxClassName<SrcImgSigT>()))
24 {
25 }
```

8.166.3.2 template<class DstImgSigT, class SrcImgSigT> HxImgFtorSet< DstImgSigT, SrcImgSigT >::~HxImgFtorSet () [virtual]

Destructor.

```
29 {
30 }
```

8.166.4 Member Function Documentation

8.166.4.1 template<class DstImgSigT, class SrcImgSigT> void HxImgFtorSet< DstImgSigT, SrcImgSigT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncSet** (p. 223) to do the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
39 {
40     HxPointInt srcBegin, dstBegin, srcEnd, dstEnd;
41
42     srcBegin = HxGetTag(tags, "srcBegin", HxPointInt(0, 0, 0));
43     srcEnd = HxGetTag(tags, "srcEnd", HxPointInt(srcSize - HxSizes(1, 1, 1)));
44     dstBegin = HxGetTag(tags, "dstBegin", HxPointInt(0, 0, 0));
45     dstEnd = dstSize - HxSizes(1, 1, 1);
46
47     if ((srcBegin.inf(HxPointInt(0, 0, 0)) != HxPointInt(0, 0, 0))
48             || (srcBegin.sup(srcEnd) != srcEnd))
49     {
50         HxEnvironment::instance()->errorStream()
51             << "Extended set: source begin out of range" << STD_ENDL;
52         return;
53     }
54
55     if ((dstBegin.inf(HxPointInt(0, 0, 0)) != HxPointInt(0, 0, 0))
56             || (dstBegin.sup(dstEnd) != dstEnd))
57     {
58         HxEnvironment::instance()->errorStream()
59             << "Extended set: destination begin out of range" << STD_ENDL;
60         return;
61     }
62
63     if ((srcEnd.inf(srcBegin) != srcBegin)
64             || (srcEnd.sup(srcSize - HxSizes(1, 1, 1))
65                 != (srcSize - HxSizes(1, 1, 1))))
66     {
67         HxEnvironment::instance()->errorStream()
68             << "Extended set: source end out of range" << STD_ENDL;
69         return;
70     }
71
72     HxSizes regionSize = HxSizes(srcEnd - srcBegin) + HxSizes(1, 1, 1);
73     regionSize = regionSize.inf(dstSize - HxSizes(dstBegin));
74
75     srcPtr.incXYZ(srcBegin.x(), srcBegin.y(), srcBegin.z());
76     dstPtr.incXYZ(dstBegin.x(), dstBegin.y(), dstBegin.z());
77
78     HxFuncSet(dstPtr, srcPtr, regionSize);
79 }
```

The documentation for this class was generated from the following files:

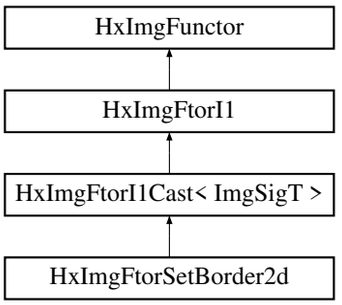
- **HxImgFtorSet.h**
- HxImgFtorSet.c

8.167 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. 147), **HxFuncBorderMirror2d** (p. 145) *(default),* **HxFuncBorderPropagate2d** (p. 149).

8.167.1 Detailed Description

template<class ImgSigT> class HxImgFtorSetBorder2d< ImgSigT >

Instantiation of generic algorithm for border operations on 2D images.

Template parameters:

- ImgSigT is the signature type of the image

8.167.2 Member Typedef Documentation

8.167.2.1 template<class ImgSigT> typedef HxImgFtorSetBorderKey HxImgFtorSetBorder2d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 732).

8.167.3 Constructor & Destructor Documentation

8.167.3.1 template<class ImgSigT> HxImgFtorSetBorder2d< ImgSigT >::HxImgFtorSetBorder2d () [inline]

Constructor.

```
20     : HxImgFtorI1Cast<ImgSigT>(
21         HxImgFtorSetBorderKey(HxClassName<ImgSigT>()))
22 {
23 }
```

8.167.3.2 template<class ImgSigT> HxImgFtorSetBorder2d< ImgSigT >::~HxImgFtorSetBorder2d () [virtual]

Destructor.

```
27 {
28 }
```

8.167.4 Member Function Documentation

8.167.4.1 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. 147), **HxFuncBorderMirror2d** (p. 145) (default), **HxFuncBorderPropagate2d** (p. 149).

Reimplemented from **HxImgFtorI1Cast** (p. 734).

```
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:

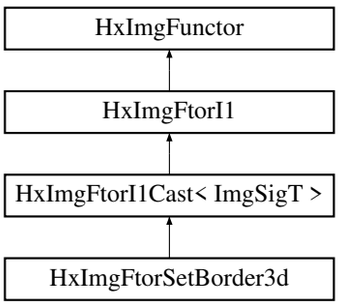
- **HxImgFtorSetBorder2d.h**
- HxImgFtorSetBorder2d.c

8.168 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, **HxImgFtorDescription** *=0)

  Calls one of the functions below to do the actual work based on the "borderType" tag: **HxFuncBorderConstant3d** (p. 148), **HxFuncBorderMirror3d** (p. 146) *(default),* **HxFuncBorderPropagate3d** (p. 150).

8.168.1 Detailed Description

template<class ImgSigT> class HxImgFtorSetBorder3d< ImgSigT >

Instantiation of generic algorithm for border operations on 3D images.

Template parameters:

- ImgSigT is the signature type of the image

8.168.2 Member Typedef Documentation

8.168.2.1 template<class ImgSigT> typedef HxImgFtorSetBorderKey HxImgFtorSetBorder3d::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI1Cast** (p. 732).

8.168.3 Constructor & Destructor Documentation

8.168.3.1 template<class ImgSigT> HxImgFtorSetBorder3d< ImgSigT >::HxImgFtorSetBorder3d () [inline]

Constructor.

```
20     : HxImgFtorI1Cast<ImgSigT>(
21         HxImgFtorSetBorderKey(HxClassName<ImgSigT>()))
22 {
23 }
```

8.168.3.2 template<class ImgSigT> HxImgFtorSetBorder3d< ImgSigT >::~HxImgFtorSetBorder3d () [virtual]

Destructor.

```
27 {
28 }
```

8.168.4 Member Function Documentation

8.168.4.1 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. 148), **HxFuncBorderMirror3d** (p. 146) (default), **HxFuncBorderPropagate3d** (p. 150).

Reimplemented from **HxImgFtorI1Cast** (p. 734).

```
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:

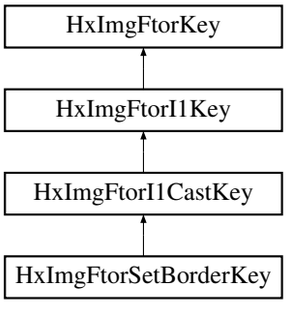
- **HxImgFtorSetBorder3d.h**
- HxImgFtorSetBorder3d.c

8.169 HxImgFtorSetBorderKey Class Reference

Key for HxImgFtorSetBorder.

`#include <HxImgFtorSetBorderKey.h>`

Inheritance diagram for HxImgFtorSetBorderKey::



Public Methods

- **HxImgFtorSetBorderKey** (**HxString** imgSig)
  Constructor.

8.169.1 Detailed Description

Key for HxImgFtorSetBorder.

8.169.2 Constructor & Destructor Documentation

8.169.2.1 HxImgFtorSetBorderKey::HxImgFtorSetBorderKey (HxString *imgSig*) [inline]

Constructor.

```
28     : HxImgFtorI1CastKey("HxImgFtorSetBorder", imgSig)
29 {
30 }
```

The documentation for this class was generated from the following file:

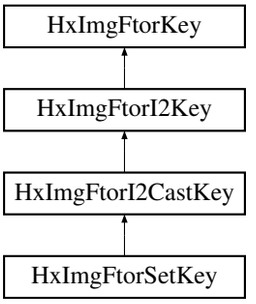
- **HxImgFtorSetBorderKey.h**

8.170 HxImgFtorSetKey Class Reference

Key for **HxImgFtorSet** (p. 847).

`#include <HxImgFtorSetKey.h>`

Inheritance diagram for HxImgFtorSetKey::



Public Methods

- **HxImgFtorSetKey** (**HxString** dstImgSig, **HxString** srcImgSig)
  Constructor.

8.170.1 Detailed Description

Key for **HxImgFtorSet** (p. 847).

8.170.2 Constructor & Destructor Documentation

8.170.2.1 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**

8.171 HxImgFtorTable Class Reference

Table with (**HxImgFtorKey** (p. 799), **HxImgFunctor** (p. 862)∗) 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** ∗)
  Add an image functor observer.
- STD_OSTREAM & **put** (STD_OSTREAM &) const
  Put on the given stream.
- ∼**HxImgFtorTable** ()
  Destructor.

Static Public Methods

- HxImgFtorTable & **instance** ()
  Access to this singleton class.

Protected Methods

- **HxImgFtorTable** ()
- **HxImgFtorTable** (const HxImgFtorTable &)

8.171.1 Detailed Description

Table with (**HxImgFtorKey** (p. 799), **HxImgFunctor** (p. 862)∗) pairs.

8.171.2 Constructor & Destructor Documentation

8.171.2.1 HxImgFtorTable::~HxImgFtorTable ()

Destructor.

```
33 {
34 #ifdef CD_TRACE
35     HxEnvironment::instance()->outputStream()
36         << "HxImgFtorTable::~HxImgFtorTable()" << STD_ENDL;
37 #endif
38 }
```

8.171.3 Member Function Documentation

8.171.3.1 HxImgFtorTable & HxImgFtorTable::instance () [static]

Access to this singleton class.

```
46 {
47     static HxImgFtorTable _instance;
48     return _instance;
49 }
```

8.171.3.2 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 }
```

8.171.3.3 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 }
```

8.171.3.4 void HxImgFtorTable::addImgFtorObserver (HxImgFtorObserver * *observer*)

Add an image functor observer.

```
95 {
96     _observerList.push_back(observer);
97 }
```

8.171.3.5 STD_OSTREAM & HxImgFtorTable::put (STD_OSTREAM & *os*) const

Put on the given stream.

```
101 {
102     Map::const_iterator i;
103     os << "HxImgFtorTable" << STD_ENDL;
104     for(i = _map.begin();i != _map.end();i++)
105     {
106         os << (*i).first << STD_ENDL;
107     }
108     return os;
109 }
```

The documentation for this class was generated from the following files:

- **HxImgFtorTable.h**
- HxImgFtorTable.c

8.172 HxImgFtorTableTem Class Template Reference

Template for typed access to the **HxImgFtorTable** (p. 856).

`#include <HxImgFtorTableTem.h>`

Public Methods

- FunctorType * **find** (const typename FunctorType::KeyType &key) const

  Find functor in the **HxImgFtorTable** (p. 856) *based on given key and cast the result to FunctorType.*

8.172.1 Detailed Description

template<class FunctorType> class HxImgFtorTableTem< FunctorType >

Template for typed access to the **HxImgFtorTable** (p. 856).

8.172.2 Member Function Documentation

8.172.2.1 template<class FunctorType> FunctorType* HxImgFtorTableTem< FunctorType >::find (const typename FunctorType::KeyType & *key*) const [inline]

Find functor in the **HxImgFtorTable** (p. 856) 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:

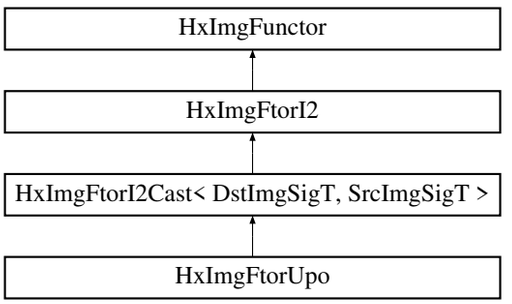
- **HxImgFtorTableTem.h**

8.173 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** (**Img1DataPtrType** dstPtr, **Img2DataPtrType** srcPtr, **HxSizes** dstSize, **HxSizes** srcSize, **HxTagList** &tags, **HxImgFtorDescription** *description=0)

  Calls **HxFuncUpoDispatch** (p. 225) *to dispatch the actual work.*

8.173.1 Detailed Description

template<class DstImgSigT, class SrcImgSigT, class UpoT> class HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >

Instantiation of generic algorithm for unary pixel operations on images.

Template parameters:

- DstImgSigT is the signature type of the destination image
- SrcImgSigT is the signature type of the source image
- UpoT is the type of the pixel functor

8.173.2 Member Typedef Documentation

8.173.2.1 template<class DstImgSigT, class SrcImgSigT, class UpoT> typedef HxImgFtorUpoKey HxImgFtorUpo::KeyType

The key type of this class.

Reimplemented from **HxImgFtorI2Cast** (p. 745).

8.173.3 Constructor & Destructor Documentation

8.173.3.1 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     HxImgFtorRuleBase::instance().setResultType(
32         HxClassName<DstImgSigT>(), "upo",
33         HxClassName<SrcImgSigT>(), HxClassName<UpoT>());
34 }
```

8.173.3.2 template<class DstImgSigT, class SrcImgSigT, class UpoT> HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >::~HxImgFtorUpo () [virtual]

Destructor.

```
38 {
39 #ifdef CD_TRACE
40     HxEnvironment::instance()->outputStream()
41         << "HxImgFtorUpo::~HxImgFtorUpo()" << STD_ENDL;
42 #endif
43 }
```

8.173.4 Member Function Documentation

8.173.4.1 template<class DstImgSigT, class SrcImgSigT, class UpoT> void HxImgFtorUpo< DstImgSigT, SrcImgSigT, UpoT >::doIt (Img1DataPtrType *dstPtr*, Img2DataPtrType *srcPtr*, HxSizes *dstSize*, HxSizes *srcSize*, HxTagList & *tags*, HxImgFtorDescription * *description* = 0) [protected, virtual]

Calls **HxFuncUpoDispatch** (p. 225) to dispatch the actual work.

Reimplemented from **HxImgFtorI2Cast** (p. 750).

```
51 {
52     UpoT upo(tags);
53
54     if (description) {
55         HxString v(typename UpoT::TransVarianceCategory().toString());
56         description->setVariation(v);
57     }
58
59     HxFuncUpoDispatch(dstPtr, srcPtr, dstSize, upo);
60 }
```

The documentation for this class was generated from the following files:

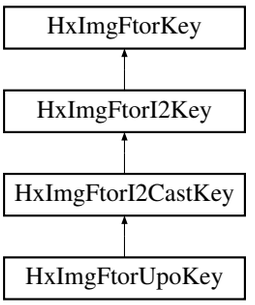
- **HxImgFtorUpo.h**
- HxImgFtorUpo.c

8.174 HxImgFtorUpoKey Class Reference

Key for **HxImgFtorUpo** (p. 859).

`#include <HxImgFtorUpoKey.h>`

Inheritance diagram for HxImgFtorUpoKey::



Public Methods

- **HxImgFtorUpoKey** (**HxString** dstImgSig, **HxString** srcImgSig, **HxString** upoName)
  Constructor.

8.174.1 Detailed Description

Key for **HxImgFtorUpo** (p. 859).

8.174.2 Constructor & Destructor Documentation

8.174.2.1 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**

8.175 HxImgFunctor Class Reference

The base class of all image functors.

`#include <HxImgFunctor.h>`

Inheritance diagram for HxImgFunctor::

Public Methods

- **HxImgFunctor** (const **HxImgFtorKey** &)
  Constructor.
- virtual ~**HxImgFunctor** ()
  Destructor.
- virtual STD_OSTREAM & **put** (STD_OSTREAM &) const
  Put on stream.
- virtual bool **probeOp** (**HxTagList** &) const
  Some operator instantiations can be queried for pre-condition information.

Protected Methods

- **HxImgFtorDescription** * **getDescription** () const

8.175.1 Detailed Description

The base class of all image functors.

8.175.2 Constructor & Destructor Documentation

8.175.2.1 HxImgFunctor::HxImgFunctor (const HxImgFtorKey & *key*)

Constructor.

```
18 {
19     HxImgFtorTable::instance().insert(key,this);
20     _key = key;
21 }
```

8.175.2.2 HxImgFunctor::~HxImgFunctor () [virtual]

Destructor.

```
24 {
25     // To do: remove from table
26 }
```

8.175.3 Member Function Documentation

8.175.3.1 STD_OSTREAM & HxImgFunctor::put (STD_OSTREAM & *os*) const [virtual]

Put on stream.

```
36 {
37     return os << _key;
38 }
```

8.175.3.2 bool HxImgFunctor::probeOp (HxTagList & *tags*) const [virtual]

Some operator instantiations can be queried for pre-condition information.

Reimplemented in **HxImgFtorKernelNgb2d** (p. 797), **HxImgFtorMNpo** (p. 805), **HxImgFtorNgb2d** (p. 812), **HxImgFtorNgb2dExtra** (p. 815), and **HxImgFtorNgb2dExtra2** (p. 818).

```
30 {
31     return true;
32 }
```

The documentation for this class was generated from the following files:

- **HxImgFunctor.h**
- HxImgFunctor.c

8.176 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.

8.176.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAbs< ImgSigT >

Instantiator for unary pixel operation with absolute value.

8.176.2 Member Data Documentation

8.176.2.1 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

8.177 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.

8.177.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorAcos**< **ImgSigT** >

Instantiator for unary pixel operation with arc cosine.

8.177.2 Member Data Documentation

8.177.2.1 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

8.178 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.

8.178.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorAdd**< **ImgSigT** >

Instantiator for binary pixel operation with addition.

8.178.2 Member Data Documentation

8.178.2.1 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

8.179 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.

8.179.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAddReduce< ImgSigT >

Instantiator for reduce operation with addition.

8.179.2 Member Data Documentation

8.179.2.1 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

8.180 HxInstantiatorAddSat Struct Template Reference

Instantiator for binary pixel operation with squared distance.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoAddSat**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.180.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorAddSat< ImgSigT >

Instantiator for binary pixel operation with squared distance.

8.180.2 Member Data Documentation

8.180.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoAddSat<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorAddSat::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxAddSat.c

8.181 HxInstantiatorAddV Class Template Reference

Instantiator for binary pixel operation with addition with a value.

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.

8.181.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAddV< ImgSigT >

Instantiator for binary pixel operation with addition with a value.

8.181.2 Member Data Documentation

8.181.2.1 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

8.182 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.

8.182.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAnd< ImgSigT >

Instantiator for binary pixel operation with and.

8.182.2 Member Data Documentation

8.182.2.1 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

8.183 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.

8.183.1 Detailed Description

template<class ImgSigT> class HxInstantiatorAndV< ImgSigT >

Instantiator for binary pixel operation with and.

8.183.2 Member Data Documentation

8.183.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.184 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.

8.184.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorArg< DstSigT, ImgSigT >

Instantiator for unary pixel operation with argument.

8.184.2 Member Data Documentation

8.184.2.1 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

8.185 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.

8.185.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorAsin**< **ImgSigT** >

Instantiator for unary pixel operation with arc sine.

8.185.2 Member Data Documentation

8.185.2.1 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

8.186 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.

8.186.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorAtan**< **ImgSigT** >

Instantiator for unary pixel operation with arc tangent.

8.186.2 Member Data Documentation

8.186.2.1 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

8.187 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.

8.187.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorAtan2< DstSigT, ImgSigT >

Instantiator for unary pixel operation with arc tangent.

8.187.2 Member Data Documentation

8.187.2.1 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

8.188 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.

8.188.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCeil< ImgSigT >

Instantiator for unary pixel operation with ceiling.

8.188.2 Member Data Documentation

8.188.2.1 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

8.189 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.

8.189.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorColSpace< DstSigT, ImgSigT >

Instantiator for unary pixel operation with color space conversion.

8.189.2 Member Data Documentation

8.189.2.1 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

8.190 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.

8.190.1 Detailed Description

template<class ImgSigT> class HxInstantiatorComplement< ImgSigT >

Instantiator for unary pixel operation with complement.

8.190.2 Member Data Documentation

8.190.2.1 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

8.191 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.

8.191.1 Detailed Description

template<class ImgSigT> class HxInstantiatorConjugate< ImgSigT >

Instantiator for unary pixel operation with conjugate.

8.191.2 Member Data Documentation

8.191.2.1 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

8.192 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.

8.192.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCos< ImgSigT >

Instantiator for unary pixel operation with cosine.

8.192.2 Member Data Documentation

8.192.2.1 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

8.193 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.

8.193.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCosh< ImgSigT >

Instantiator for unary pixel operation with hyperbolic cosine.

8.193.2 Member Data Documentation

8.193.2.1 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

8.194 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.

8.194.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCross< ImgSigT >

Instantiator for binary pixel operation with cross product.

8.194.2 Member Data Documentation

8.194.2.1 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

8.195 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.

8.195.1 Detailed Description

template<class ImgSigT> class HxInstantiatorCrossV< ImgSigT >

Instantiator for binary pixel operation with cross product.

8.195.2 Member Data Documentation

8.195.2.1 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

8.196 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.

8.196.1 Detailed Description

template<class ImgSigT> class HxInstantiatorDiv< ImgSigT >

Instantiator for binary pixel operation with division.

8.196.2 Member Data Documentation

8.196.2.1 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

8.197 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.

8.197.1 Detailed Description

template<class ImgSigT> class HxInstantiatorDivV< ImgSigT >

Instantiator for binary pixel operation with division.

8.197.2 Member Data Documentation

8.197.2.1 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

8.198 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.

8.198.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorDot< DstSigT, ImgSigT >

Instantiator for binary pixel operation with dot product.

8.198.2 Member Data Documentation

8.198.2.1 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

8.199 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.

8.199.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorDotV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with dot product.

8.199.2 Member Data Documentation

8.199.2.1 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

8.200 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.

8.200.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with equal.

8.200.2 Member Data Documentation

8.200.2.1 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

8.201 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.

8.201.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with equal.

8.201.2 Member Data Documentation

8.201.2.1 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

8.202 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.

8.202.1 Detailed Description

template<class ImgSigT> class HxInstantiatorExp< ImgSigT >

Instantiator for unary pixel operation with exponent.

8.202.2 Member Data Documentation

8.202.2.1 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

8.203 HxInstantiatorExpPix Class Template Reference

Instantiator for export operation to native arrays.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxExportPix< typename ImgSigT::ArithType, DataT > > **f**

Instantiate image functor.

8.203.1 Detailed Description

template<class ImgSigT, class DataT> class HxInstantiatorExpPix< ImgSigT, DataT >

Instantiator for export operation to native arrays.

8.203.2 Member Data Documentation

8.203.2.1 template<class ImgSigT, class DataT> HxImgFtorInOut<ImgSigT, HxExportPix<typename ImgSigT::ArithType, DataT> > HxInstantiatorExpPix::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxExportPixInst.c

8.204 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.

8.204.1 Detailed Description

template<class ImgSigT> class HxInstantiatorFloor< ImgSigT >

Instantiator for unary pixel operation with floor.

8.204.2 Member Data Documentation

8.204.2.1 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

8.205 HxInstantiatorGpi Class Template Reference

Instantiator for unary pixel operation with get pixel element.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxUpoGetPixElt< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.205.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGpi< DstSigT, ImgSigT >

Instantiator for unary pixel operation with get pixel element.

8.205.2 Member Data Documentation

8.205.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoGetPixElt<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorGpi::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoGetPixEltInst.c

8.206 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.

8.206.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater equal.

8.206.2 Member Data Documentation

8.206.2.1 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

8.207 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.

8.207.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater equal.

8.207.2 Member Data Documentation

8.207.2.1 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

8.208 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.

8.208.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterThan< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater than.

8.208.2 Member Data Documentation

8.208.2.1 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

8.209 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.

8.209.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorGreaterThanV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with greater than.

8.209.2 Member Data Documentation

8.209.2.1 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

8.210 HxInstantiatorHighlightRegion Class Template Reference

Instantiator for binary pixel operation with region highlighting.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, LblSigT, **HxBpoHighlightRegion**< typename ImgSigT::ArithType, typename ImgSigT::ArithTypeDouble, typename LblSigT::ArithType > > **f**

  Instantiate image functor.

8.210.1 Detailed Description

template<class ImgSigT, class LblSigT> class HxInstantiatorHighlightRegion< ImgSigT, LblSigT >

Instantiator for binary pixel operation with region highlighting.

8.210.2 Member Data Documentation

8.210.2.1 template<class ImgSigT, class LblSigT> HxImgFtorBpo<ImgSigT, ImgSigT, LblSigT, HxBpoHighlightRegion<typename ImgSigT::ArithType, typename ImgSigT::ArithTypeDouble, typename LblSigT::ArithType> > HxInstantiatorHighlightRegion::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxHighlightRegion.c

8.211 HxInstantiatorImpPix Class Template Reference

Instantiator for import operation from native array.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxImportPix< typename ImgSigT::ArithType, DataT > > **f**
  Instantiate image functor.

8.211.1 Detailed Description

template<class ImgSigT, class DataT> class HxInstantiatorImpPix< ImgSigT, DataT >

Instantiator for import operation from native array.

8.211.2 Member Data Documentation

8.211.2.1 template<class ImgSigT, class DataT> HxImgFtorInOut<ImgSigT, HxImportPix<typename ImgSigT::ArithType, DataT> > HxInstantiatorImpPix::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxImportPixInst.c

8.212 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.

8.212.1 Detailed Description

template<class ImgSigT> class HxInstantiatorInf< ImgSigT >

Instantiator for binary pixel operation with infimum.

8.212.2 Member Data Documentation

8.212.2.1 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

8.213 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.

8.213.1 Detailed Description

template<class ImgSigT> class HxInstantiatorInfReduce< ImgSigT >

Instantiator for reduce operation with infimum.

8.213.2 Member Data Documentation

8.213.2.1 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

8.214 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.

8.214.1 Detailed Description

template<class ImgSigT> class HxInstantiatorInfV< ImgSigT >

Instantiator for binary pixel operation with infimum.

8.214.2 Member Data Documentation

8.214.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.215 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.

8.215.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLeftShift< ImgSigT >

Instantiator for binary pixel operation with left shift.

8.215.2 Member Data Documentation

8.215.2.1 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

8.216 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.

8.216.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorLeftShiftV**< **ImgSigT** >

Instantiator for binary pixel operation with left shift.

8.216.2 Member Data Documentation

8.216.2.1 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

8.217 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.

8.217.1 Detailed Description

template<**class DstSigT, class ImgSigT**> **class HxInstantiatorLessEqual**< **DstSigT, ImgSigT** >

Instantiator for binary pixel operation with less equal.

8.217.2 Member Data Documentation

8.217.2.1 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

8.218 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.

8.218.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less equal.

8.218.2 Member Data Documentation

8.218.2.1 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

8.219 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.

8.219.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessThan< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less than.

8.219.2 Member Data Documentation

8.219.2.1 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

8.220 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.

8.220.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorLessThanV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with less than.

8.220.2 Member Data Documentation

8.220.2.1 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

8.221 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.

8.221.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLog< ImgSigT >

Instantiator for unary pixel operation with natural logarithm.

8.221.2 Member Data Documentation

8.221.2.1 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

8.222 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.

8.222.1 Detailed Description

template<class ImgSigT> class HxInstantiatorLog10< ImgSigT >

Instantiator for unary pixel operation with base 10 logarithm.

8.222.2 Member Data Documentation

8.222.2.1 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

8.223 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.

8.223.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMax< ImgSigT >

Instantiator for binary pixel operation with maximum.

8.223.2 Member Data Documentation

8.223.2.1 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

8.224 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.

8.224.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMaxReduce< ImgSigT >

Instantiator for reduce operation with maximum.

8.224.2 Member Data Documentation

8.224.2.1 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

8.225 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.

8.225.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMaxV< ImgSigT >

Instantiator for binary pixel operation with maximum.

8.225.2 Member Data Documentation

8.225.2.1 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

8.226 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.

8.226.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMin< ImgSigT >

Instantiator for binary pixel operation with minimum.

8.226.2 Member Data Documentation

8.226.2.1 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

8.227 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.

8.227.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorMinReduce**< **ImgSigT** >

Instantiator for reduce operation with minimum.

8.227.2 Member Data Documentation

8.227.2.1 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

8.228 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.

8.228.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorMinV**< **ImgSigT** >

Instantiator for binary pixel operation with minimum.

8.228.2 Member Data Documentation

8.228.2.1 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

8.229 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.

8.229.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMod< ImgSigT >

Instantiator for binary pixel operation with modulo.

8.229.2 Member Data Documentation

8.229.2.1 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

8.230 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.

8.230.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorModV**< **ImgSigT** >

Instantiator for binary pixel operation with modulo.

8.230.2 Member Data Documentation

8.230.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.231 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.

8.231.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorMul**< **ImgSigT** >

Instantiator for binary pixel operation with multiplication.

8.231.2 Member Data Documentation

8.231.2.1 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

8.232 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.

8.232.1 Detailed Description

template<class ImgSigT> class HxInstantiatorMulReduce< ImgSigT >

Instantiator for reduce operation with multiplication.

8.232.2 Member Data Documentation

8.232.2.1 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

8.233 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.

8.233.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorMulV**< **ImgSigT** >

Instantiator for binary pixel operation with multiplication.

8.233.2 Member Data Documentation

8.233.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.234 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.

8.234.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorNegate**< **ImgSigT** >

Instantiator for unary pixel operation with negation.

8.234.2 Member Data Documentation

8.234.2.1 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

8.235 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.

8.235.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNorm1< DstSigT, ImgSigT >

Instantiator for unary pixel operation with L1 norm.

8.235.2 Member Data Documentation

8.235.2.1 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

8.236 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.

8.236.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNorm2< DstSigT, ImgSigT >

Instantiator for unary pixel operation with L2 norm.

8.236.2 Member Data Documentation

8.236.2.1 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

8.237 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.

8.237.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNorm2Sqr< DstSigT, ImgSigT >

Instantiator for unary pixel operation with squared L2 norm.

8.237.2 Member Data Documentation

8.237.2.1 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

8.238 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.

8.238.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNormInf< DstSigT, ImgSigT >

Instantiator for unary pixel operation with L inf norm.

8.238.2 Member Data Documentation

8.238.2.1 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

8.239 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.

8.239.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNotEqual< DstSigT, ImgSigT >

Instantiator for binary pixel operation with not equal.

8.239.2 Member Data Documentation

8.239.2.1 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

8.240 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.

8.240.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorNotEqualV< DstSigT, ImgSigT >

Instantiator for binary pixel operation with not equal.

8.240.2 Member Data Documentation

8.240.2.1 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

8.241 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.

8.241.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorOr**< **ImgSigT** >

Instantiator for binary pixel operation with or.

8.241.2 Member Data Documentation

8.241.2.1 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

8.242 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.

8.242.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorOrV**< **ImgSigT** >

Instantiator for binary pixel operation with or.

8.242.2 Member Data Documentation

8.242.2.1 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

8.243 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.

8.243.1 Detailed Description

template<class ImgSigT> class HxInstantiatorPow< ImgSigT >

Instantiator for binary pixel operation with power.

8.243.2 Member Data Documentation

8.243.2.1 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

8.244 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.

8.244.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorPowV**< **ImgSigT** >

Instantiator for binary pixel operation with power.

8.244.2 Member Data Documentation

8.244.2.1 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

8.245 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.

8.245.1 Detailed Description

template<**class DstSigT, class ImgSigT**> **class HxInstantiatorProduct**< **DstSigT, ImgSigT** >

Instantiator for unary pixel operation with product.

8.245.2 Member Data Documentation

8.245.2.1 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

8.246 HxInstantiatorRGB2Intensity Class Template Reference

Instantiator for unary pixel operation with **RGB2Intensity** (p. 1361).

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, **RGB2Intensity**< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.246.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorRGB2Intensity< DstSigT, ImgSigT >

Instantiator for unary pixel operation with **RGB2Intensity** (p. 1361).

8.246.2 Member Data Documentation

8.246.2.1 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

8.247 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.

8.247.1 Detailed Description

template<class ImgSigT> class HxInstantiatorRightShift< ImgSigT >

Instantiator for binary pixel operation with right shift.

8.247.2 Member Data Documentation

8.247.2.1 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

8.248 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.

8.248.1 Detailed Description

template<class ImgSigT> class HxInstantiatorRightShiftV< ImgSigT >

Instantiator for binary pixel operation with right shift.

8.248.2 Member Data Documentation

8.248.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.249 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.

8.249.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorRound**< **ImgSigT** >

Instantiator for unary pixel operation with round.

8.249.2 Member Data Documentation

8.249.2.1 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

8.250 HxInstantiatorSet Struct Template Reference

Instantiator for set operation.

`#include <HxImgFtorSetInst.h>`

Public Attributes

- **HxImgFtorSet**< **HxImageSig2dByte**, SrcSigT > **f001**

  Instantiate image functor for one image type.

- **HxImgFtorSet**< **HxImageSig2dShort**, SrcSigT > **f002**

  Instantiate image functor for one image type.

- **HxImgFtorSet**< **HxImageSig2dInt**, SrcSigT > **f003**

  Instantiate image functor for one image type.

- **HxImgFtorSet**< **HxImageSig2dFloat**, SrcSigT > **f004**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dDouble**, SrcSigT > **f005**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec2Byte**, SrcSigT > **f006**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec2Short**, SrcSigT > **f007**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec2Int**, SrcSigT > **f008**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec2Float**, SrcSigT > **f009**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec2Double**, SrcSigT > **f010**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec3Byte**, SrcSigT > **f011**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec3Short**, SrcSigT > **f012**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec3Int**, SrcSigT > **f013**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec3Float**, SrcSigT > **f014**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig2dVec3Double**, SrcSigT > **f015**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig3dByte**, SrcSigT > **f016**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig3dShort**, SrcSigT > **f017**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig3dInt**, SrcSigT > **f018**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig3dFloat**, SrcSigT > **f019**
  Instantiate image functor for one image type.
- **HxImgFtorSet**< **HxImageSig3dDouble**, SrcSigT > **f020**

Instantiate image functor for one image type.

- **HxImgFtorSet< HxImageSig2dComplex**, SrcSigT > **f021**
  Instantiate image functor for one image type.

8.250.1 Detailed Description

template<class SrcSigT> struct HxInstantiatorSet< SrcSigT >

Instantiator for set operation.

8.250.2 Member Data Documentation

8.250.2.1 template<class SrcSigT> HxImgFtorSet<HxImageSig2dByte, SrcSigT> HxInstantiatorSet::f001

Instantiate image functor for one image type.

8.250.2.2 template<class SrcSigT> HxImgFtorSet<HxImageSig2dShort, SrcSigT> HxInstantiatorSet::f002

Instantiate image functor for one image type.

8.250.2.3 template<class SrcSigT> HxImgFtorSet<HxImageSig2dInt, SrcSigT> HxInstantiatorSet::f003

Instantiate image functor for one image type.

8.250.2.4 template<class SrcSigT> HxImgFtorSet<HxImageSig2dFloat, SrcSigT> HxInstantiatorSet::f004

Instantiate image functor for one image type.

8.250.2.5 template<class SrcSigT> HxImgFtorSet<HxImageSig2dDouble, SrcSigT> HxInstantiatorSet::f005

Instantiate image functor for one image type.

8.250.2.6 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec2Byte, SrcSigT> HxInstantiatorSet::f006

Instantiate image functor for one image type.

8.250.2.7 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec2Short, SrcSigT> HxInstantiatorSet::f007

Instantiate image functor for one image type.

8.250.2.8 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec2Int, SrcSigT> HxInstantiatorSet::f008

Instantiate image functor for one image type.

8.250.2.9 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec2Float, SrcSigT> HxInstantiatorSet::f009

Instantiate image functor for one image type.

8.250.2.10 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec2Double, SrcSigT> HxInstantiatorSet::f010

Instantiate image functor for one image type.

8.250.2.11 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec3Byte, SrcSigT> HxInstantiatorSet::f011

Instantiate image functor for one image type.

8.250.2.12 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec3Short, SrcSigT> HxInstantiatorSet::f012

Instantiate image functor for one image type.

8.250.2.13 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec3Int, SrcSigT> HxInstantiatorSet::f013

Instantiate image functor for one image type.

8.250.2.14 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec3Float, SrcSigT> HxInstantiatorSet::f014

Instantiate image functor for one image type.

8.250.2.15 template<class SrcSigT> HxImgFtorSet<HxImageSig2dVec3Double, SrcSigT> HxInstantiatorSet::f015

Instantiate image functor for one image type.

8.250.2.16 template<class SrcSigT> HxImgFtorSet<HxImageSig3dByte, SrcSigT> HxInstantiatorSet::f016

Instantiate image functor for one image type.

8.250.2.17 template<class SrcSigT> HxImgFtorSet<HxImageSig3dShort, SrcSigT> HxInstantiatorSet::f017

Instantiate image functor for one image type.

8.250.2.18 template<class SrcSigT> HxImgFtorSet<HxImageSig3dInt, SrcSigT> HxInstantiatorSet::f018

Instantiate image functor for one image type.

8.250.2.19 template<class SrcSigT> HxImgFtorSet<HxImageSig3dFloat, SrcSigT> HxInstantiatorSet::f019

Instantiate image functor for one image type.

8.250.2.20 template<class SrcSigT> HxImgFtorSet<HxImageSig3dDouble, SrcSigT> HxInstantiatorSet::f020

Instantiate image functor for one image type.

8.250.2.21 template<class SrcSigT> HxImgFtorSet<HxImageSig2dComplex, SrcSigT> HxInstantiatorSet::f021

Instantiate image functor for one image type.

The documentation for this struct was generated from the following file:

- **HxImgFtorSetInst.h**

8.251 HxInstantiatorSetPartImg Struct Template Reference

Instantiator for unary pixel operation.

Public Attributes

- **HxImgFtorUpo**< ImgSigT, ImgSigT, HxUpoSetPartImage< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.251.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorSetPartImg< ImgSigT >

Instantiator for unary pixel operation.

8.251.2 Member Data Documentation

8.251.2.1 template<class ImgSigT> HxImgFtorUpo< ImgSigT, ImgSigT, HxUpoSetPartImage<typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSetPartImg::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- **HxUpoSetPartImageInst.c**

8.252 HxInstantiatorSetVal Class Template Reference

Instantiator for inout operation to set values.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxInOutSetVal< typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.252.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSetVal< ImgSigT >

Instantiator for inout operation to set values.

8.252.2 Member Data Documentation

8.252.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxInOutSetVal<typename ImgSigT::ArithType> > HxInstantiatorSetVal::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxInOutSetValInst.c

8.253 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.

8.253.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSin< ImgSigT >

Instantiator for unary pixel operation with sine.

8.253.2 Member Data Documentation

8.253.2.1 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

8.254 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.

8.254.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSinh< ImgSigT >

Instantiator for unary pixel operation with hyperbolic sine.

8.254.2 Member Data Documentation

8.254.2.1 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

8.255 HxInstantiatorSpi Class Template Reference

Instantiator for binary pixel operation with set pixel element.

Public Attributes

- **HxImgFtorBpo**< DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt< typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.255.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorSpi< DstSigT, ImgSigT >

Instantiator for binary pixel operation with set pixel element.

8.255.2 Member Data Documentation

8.255.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, DstSigT, ImgSigT, HxBpoSetPixElt<typename DstSigT::ArithType, typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSpi::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoSetPixEltInst.c

8.256 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.

8.256.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorSqrDst< ImgSigT >

Instantiator for binary pixel operation with squared distance.

8.256.2 Member Data Documentation

8.256.2.1 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

8.257 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.

8.257.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorSqrt**< **ImgSigT** >

Instantiator for unary pixel operation with square root.

8.257.2 Member Data Documentation

8.257.2.1 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

8.258 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.

8.258.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSub< ImgSigT >

Instantiator for binary pixel operation with subtraction.

8.258.2 Member Data Documentation

8.258.2.1 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

8.259 HxInstantiatorSubSat Struct Template Reference

Instantiator for binary pixel operation with squared distance.

Public Attributes

- **HxImgFtorBpo**< ImgSigT, ImgSigT, ImgSigT, **HxBpoSubSat**< typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.259.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorSubSat< ImgSigT >

Instantiator for binary pixel operation with squared distance.

8.259.2 Member Data Documentation

8.259.2.1 template<class ImgSigT> HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT, HxBpoSubSat<typename ImgSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorSubSat::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxSubSat.c

8.260 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.

8.260.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorSubV**< **ImgSigT** >

Instantiator for binary pixel operation with subtraction.

8.260.2 Member Data Documentation

8.260.2.1 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

8.261 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.

8.261.1 Detailed Description

template<**class DstSigT, class ImgSigT**> **class HxInstantiatorSum**< **DstSigT, ImgSigT** >

Instantiator for unary pixel operation with sum.

8.261.2 Member Data Documentation

8.261.2.1 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

8.262 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.

8.262.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSup< ImgSigT >

Instantiator for binary pixel operation with supremum.

8.262.2 Member Data Documentation

8.262.2.1 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

8.263 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.

8.263.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSupReduce< ImgSigT >

Instantiator for reduce operation with supremum.

8.263.2 Member Data Documentation

8.263.2.1 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

8.264 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.

8.264.1 Detailed Description

template<class ImgSigT> class HxInstantiatorSupV< ImgSigT >

Instantiator for binary pixel operation with supremum.

8.264.2 Member Data Documentation

8.264.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.265 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.

8.265.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorTan**< **ImgSigT** >

Instantiator for unary pixel operation with tangent.

8.265.2 Member Data Documentation

8.265.2.1 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

8.266 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.

8.266.1 Detailed Description

template<**class ImgSigT**> **class HxInstantiatorTanh**< **ImgSigT** >

Instantiator for unary pixel operation with hyperbolic tangent.

8.266.2 Member Data Documentation

8.266.2.1 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

8.267 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.

8.267.1 Detailed Description

template<class ImgSigT> struct HxInstantiatorTriStateThreshold< ImgSigT >

Instantiator for unary pixel operation with tri state threshold.

8.267.2 Member Data Documentation

8.267.2.1 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

8.268 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.

8.268.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorUpoMax< DstSigT, ImgSigT >

Instantiator for unary pixel operation with unary maximum.

8.268.2 Member Data Documentation

8.268.2.1 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

8.269 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.

8.269.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorUpoMin< DstSigT, ImgSigT >

Instantiator for unary pixel operation with unary minimum.

8.269.2 Member Data Documentation

8.269.2.1 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

8.270 HxInstantiatorUpoThreshold Class Template Reference

Instantiator for unary pixel operation with threshold.

Public Attributes

- **HxImgFtorUpo**< DstSigT, ImgSigT, HxUpoThreshold< typename DstSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.270.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorUpoThreshold< DstSigT, ImgSigT >

Instantiator for unary pixel operation with threshold.

8.270.2 Member Data Documentation

8.270.2.1 template<class DstSigT, class ImgSigT> HxImgFtorUpo<DstSigT, ImgSigT, HxUpoThreshold<typename DstSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorUpoThreshold::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxUpoThresholdInst.c

8.271 HxInstantiatorVec2 Class Template Reference

Instantiator for binary pixel operation with vec2.

Public Attributes

- **HxImgFtorBpo**< DstSigT, ImgSigT, ImgSigT, HxBpoVec2< typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.271.1 Detailed Description

template<class DstSigT, class ImgSigT> class HxInstantiatorVec2< DstSigT, ImgSigT >

Instantiator for binary pixel operation with vec2.

8.271.2 Member Data Documentation

8.271.2.1 template<class DstSigT, class ImgSigT> HxImgFtorBpo<DstSigT, ImgSigT, ImgSigT, HxBpoVec2<typename DstSigT::ArithType, typename ImgSigT::ArithType, typename ImgSigT::ArithType> > HxInstantiatorVec2::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxBpoVec2Inst.c

8.272 HxInstantiatorVec3 Class Template Reference

Instantiator for multi pixel operation with vec3.

Public Attributes

- **HxImgFtorMpo**< DstSigT, SrcsSigT, HxMpoVec3< typename DstSigT::ArithType, typename SrcsSigT::ArithType > > **f**

  Instantiate image functor.

8.272.1 Detailed Description

template<class DstSigT, class SrcsSigT> class HxInstantiatorVec3< DstSigT, SrcsSigT >

Instantiator for multi pixel operation with vec3.

8.272.2 Member Data Documentation

8.272.2.1 template<class DstSigT, class SrcsSigT> HxImgFtorMpo<DstSigT, SrcsSigT, HxMpoVec3<typename DstSigT::ArithType, typename SrcsSigT::ArithType> > HxInstantiatorVec3::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxMpoVec3Inst.c

8.273 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.

8.273.1 Detailed Description

template<class ImgSigT> class HxInstantiatorXor< ImgSigT >

Instantiator for binary pixel operation with exclusive or.

8.273.2 Member Data Documentation

8.273.2.1 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

8.274 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.

8.274.1 Detailed Description

template<class ImgSigT> class HxInstantiatorXorV< ImgSigT >

Instantiator for binary pixel operation with exclusive or.

8.274.2 Member Data Documentation

8.274.2.1 template<class ImgSigT> HxImgFtorUpo<ImgSigT, ImgSigT, HxBpoBind2Val<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

8.275 HxInstDiyTranspose Class Template Reference

Instantiator for DIY operation with transpose.

Public Attributes

- **HxImgFtorDiy**< ImgSigT, ImgSigT, **HxDiyTranspose**< typename ImgSigT::DataPtrType, typename ImgSigT::DataPtrType > > **f**

  Instantiate image functor.

8.275.1 Detailed Description

template<**class ImgSigT**> **class HxInstDiyTranspose**< **ImgSigT** >

Instantiator for DIY operation with transpose.

8.275.2 Member Data Documentation

8.275.2.1 template<class ImgSigT> HxImgFtorDiy< ImgSigT, ImgSigT, HxDiyTranspose<typename ImgSigT::DataPtrType, typename ImgSigT::DataPtrType> > HxInstDiyTranspose::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- **HxTranspose.c**

8.276 HxInstExportExtraIdentMaskCentralMoments Struct Template Reference

Instantiator for export extra operation with identification mask central moments.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraIdentMaskCentralMoments**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

  Instantiate image functor.

8.276.1 Detailed Description

template<class ImgSigT, class ExtraImSigT> struct HxInstExportExtraIdentMaskCentralMoments< ImgSigT, ExtraImSigT >

Instantiator for export extra operation with identification mask central moments.

8.276.2 Member Data Documentation

8.276.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskCentralMoments<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraIdentMaskCentralMoments::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraIdentMaskCentralMomentsInst.c

8.277 HxInstExportExtraIdentMaskMean Struct Template Reference

Instantiator for export extra operation with identification mask mean.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraIdentMaskMean**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

  Instantiate image functor.

8.277.1 Detailed Description

template<class ImgSigT, class ExtraImSigT> struct HxInstExportExtraIdentMaskMean< ImgSigT, ExtraImSigT >

Instantiator for export extra operation with identification mask mean.

8.277.2 Member Data Documentation

8.277.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMean<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraIdentMaskMean::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraIdentMaskMeanInst.c

8.278 HxInstExportExtraIdentMaskMedian Struct Template Reference

Instantiator for export extra operation with identification mask median.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraIdentMaskMedian**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

  Instantiate image functor.

8.278.1 Detailed Description

template<class ImgSigT, class ExtraImSigT> struct HxInstExportExtraIdentMaskMedian< ImgSigT, ExtraImSigT >

Instantiator for export extra operation with identification mask median.

8.278.2 Member Data Documentation

8.278.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMedian<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraIdentMaskMedian::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraIdentMaskMedianInst.c

8.279 HxInstExportExtraIdentMaskMoments Struct Template Reference

Instantiator for export extra operation with identification mask moments.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraIdentMaskMoments**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

Instantiate image functor.

8.279.1 Detailed Description

template<class ImgSigT, class ExtraImSigT> struct HxInstExportExtraIdentMaskMoments< ImgSigT, ExtraImSigT >

Instantiator for export extra operation with identification mask moments.

8.279.2 Member Data Documentation

8.279.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskMoments<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraIdentMaskMoments::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraIdentMaskMomentsInst.c

8.280 HxInstExportExtraIdentMaskStdev Struct Template Reference

Instantiator for export extra operation with identification mask standard deviation.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraIdentMaskStdev**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

  Instantiate image functor.

8.280.1 Detailed Description

template<class ImgSigT, class ExtraImSigT> struct HxInstExportExtraIdentMaskStdev< ImgSigT, ExtraImSigT >

Instantiator for export extra operation with identification mask standard deviation.

8.280.2 Member Data Documentation

8.280.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskStdev<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraIdentMaskStdev::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraIdentMaskStdevInst.c

8.281 HxInstExportExtraIdentMaskSum Struct Template Reference

Instantiator for export extra operation with identification mask summation.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraIdentMaskSum**< typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

  Instantiate image functor.

8.281.1 Detailed Description

template<class ImgSigT, class ExtraImSigT> struct HxInstExportExtraIdentMaskSum< ImgSigT, ExtraImSigT >

Instantiator for export extra operation with identification mask summation.

8.281.2 Member Data Documentation

8.281.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraIdentMaskSum<typename ImgSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraIdentMaskSum::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraIdentMaskSumInst.c

8.282 HxInstExportExtraWeightMaskSum Struct Template Reference

Instantiator for export extra operation with weight mask summation.

Public Attributes

- **HxImgFtorExportExtra**< ImgSigT, ExtraImSigT, **HxExportExtraWeightMaskSum**< typename ExtraImSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > **f**

  Instantiate image functor.

8.282.1 Detailed Description

template<**class ImgSigT, class ExtraImSigT**> **struct HxInstExportExtraWeightMaskSum**< **ImgSigT, ExtraImSigT** >

Instantiator for export extra operation with weight mask summation.

8.282.2 Member Data Documentation

8.282.2.1 template<class ImgSigT, class ExtraImSigT> HxImgFtorExportExtra< ImgSigT, ExtraImSigT, HxExportExtraWeightMaskSum<typename ExtraImSigT::ArithTypeDouble, typename ImgSigT::ArithType, typename ExtraImSigT::ArithType > > HxInstExportExtraWeightMaskSum::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxExportExtraWeightMaskSumInst.c

8.283 HxInstExpPpm Class Template Reference

Instantiator for export operation to ppm format.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxExportPpm< typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.283.1 Detailed Description

template<**class ImgSigT**> **class HxInstExpPpm**< **ImgSigT** >

Instantiator for export operation to ppm format.

8.283.2 Member Data Documentation

8.283.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxExportPpm<typename ImgSigT::ArithType> > HxInstExpPpm::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxExportPpmInst.c

8.284 HxInstGenConv2dAddInf 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.

8.284.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dAddInf< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and infimum as basic operations using a 2d kernel.

8.284.2 Member Data Documentation

8.284.2.1 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> > HxInstGenConv2dAddInf::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddInfInst.c

8.285 HxInstGenConv2dAddMax 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.

8.285.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dAddMax< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and maximum as basic operations using a 2d kernel.

8.285.2 Member Data Documentation

8.285.2.1 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> > HxInstGenConv2dAddMax::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMaxInst.c

8.286 HxInstGenConv2dAddMin 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.

8.286.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dAddMin< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and minimum as basic operations using a 2d kernel.

8.286.2 Member Data Documentation

8.286.2.1 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> > HxInstGenConv2dAddMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMinInst.c

8.287 HxInstGenConv2dAddSup 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.

8.287.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dAddSup< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and supremum as basic operations using a 2d kernel.

8.287.2 Member Data Documentation

8.287.2.1 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> > HxInstGenConv2dAddSup::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddSupInst.c

8.288 HxInstGenConv2dK1dAddInf 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.

8.288.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dK1dAddInf< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and infimum as basic operations using a 1d kernel.

8.288.2 Member Data Documentation

8.288.2.1 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> > HxInstGenConv2dK1dAddInf::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddInfInst.c

8.289 HxInstGenConv2dK1dAddMax 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.

8.289.1 Detailed Description

template<**class ImgSigT, class KerSigT**> **class HxInstGenConv2dK1dAddMax**< **ImgSigT, KerSigT** >

Instantiator for generalized convolution operation on 2d images with addition and maximum as basic operations using a 1d kernel.

8.289.2 Member Data Documentation

8.289.2.1 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> > HxInstGenConv2dK1dAddMax::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMaxInst.c

8.290 HxInstGenConv2dK1dAddMin 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.

8.290.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dK1dAddMin< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and minimum as basic operations using a 1d kernel.

8.290.2 Member Data Documentation

8.290.2.1 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> > HxInstGenConv2dK1dAddMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMinInst.c

8.291 HxInstGenConv2dK1dAddSup 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.

8.291.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dK1dAddSup< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with addition and supremum as basic operations using a 1d kernel.

8.291.2 Member Data Documentation

8.291.2.1 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> > HxInstGenConv2dK1dAddSup::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddSupInst.c

8.292 HxInstGenConv2dK1dMulAdd 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.

8.292.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dK1dMulAdd< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with multiplication and addition as basic operations using a 1d kernel.

8.292.2 Member Data Documentation

8.292.2.1 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> > HxInstGenConv2dK1dMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

8.293 HxInstGenConv2dMulAdd 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.

8.293.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dMulAdd< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 2d images with multiplication and addition as basic operations using a 2d kernel.

8.293.2 Member Data Documentation

8.293.2.1 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> > HxInstGenConv2dMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

8.294 HxInstGenConv2dSepAddInf Class Template Reference

Instantiator for separable generalized convolution operation on 2d images with addition and infimum as basic operations using two 1d kernels.

Public Attributes

- **HxImgFtorGenConv2dSep**< 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.

8.294.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dSepAddInf< ImgSigT, KerSigT >

Instantiator for separable generalized convolution operation on 2d images with addition and infimum as basic operations using two 1d kernels.

8.294.2 Member Data Documentation

8.294.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dSep< 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> > HxInstGenConv2dSepAddInf::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddInfInst.c

8.295 HxInstGenConv2dSepAddMax Class Template Reference

Instantiator for separable generalized convolution operation on 2d images with addition and maximum as basic operations using two 1d kernels.

Public Attributes

- **HxImgFtorGenConv2dSep**< 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.

8.295.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dSepAddMax< ImgSigT, KerSigT >

Instantiator for separable generalized convolution operation on 2d images with addition and maximum as basic operations using two 1d kernels.

8.295.2 Member Data Documentation

8.295.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dSep< 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> > HxInstGenConv2dSepAddMax::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMaxInst.c

8.296 HxInstGenConv2dSepAddMin Class Template Reference

Instantiator for separable generalized convolution operation on 2d images with addition and minimum as basic operations using two 1d kernels.

Public Attributes

- **HxImgFtorGenConv2dSep**< 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.

8.296.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dSepAddMin< ImgSigT, KerSigT >

Instantiator for separable generalized convolution operation on 2d images with addition and minimum as basic operations using two 1d kernels.

8.296.2 Member Data Documentation

8.296.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dSep< 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> > HxInstGenConv2dSepAddMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddMinInst.c

8.297 HxInstGenConv2dSepAddSup Class Template Reference

Instantiator for separable generalized convolution operation on 2d images with addition and supremum as basic operations using two 1d kernels.

Public Attributes

- **HxImgFtorGenConv2dSep**< 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.

8.297.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dSepAddSup< ImgSigT, KerSigT >

Instantiator for separable generalized convolution operation on 2d images with addition and supremum as basic operations using two 1d kernels.

8.297.2 Member Data Documentation

8.297.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dSep< 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> > HxInstGenConv2dSepAddSup::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvAddSupInst.c

8.298 HxInstGenConv2dSepMulAdd Class Template Reference

Instantiator for separable generalized convolution operation on 2d images with multiplication and addition as basic operations using two 1d kernels.

Public Attributes

- **HxImgFtorGenConv2dSep**< 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.

8.298.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv2dSepMulAdd< ImgSigT, KerSigT >

Instantiator for separable generalized convolution operation on 2d images with multiplication and addition as basic operations using two 1d kernels.

8.298.2 Member Data Documentation

8.298.2.1 template<class ImgSigT, class KerSigT> HxImgFtorGenConv2dSep< 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> > HxInstGenConv2dSepMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

8.299 HxInstGenConv3dK1dMulAdd 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.

8.299.1 Detailed Description

template<class DstSigT, class SrcSigT, class KerSigT> class HxInstGenConv3dK1dMulAdd< DstSigT, SrcSigT, KerSigT >

Instantiator for generalized convolution operation on 3d images with multiplication and addition as basic operations using a 1d kernel.

8.299.2 Member Data Documentation

8.299.2.1 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> > HxInstGenConv3dK1dMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

8.300 HxInstGenConv3dMulAdd 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.

8.300.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstGenConv3dMulAdd< ImgSigT, KerSigT >

Instantiator for generalized convolution operation on 3d images with multiplication and addition as basic operations using a 3d kernel.

8.300.2 Member Data Documentation

8.300.2.1 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> > HxInstGenConv3dMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGenConvMulAddInst.c

8.301 HxInstGeneratePix Class Template Reference

Instantiator for inout operation to generate pixels.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxGeneratePix< typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.301.1 Detailed Description

template<class ImgSigT> class HxInstGeneratePix< ImgSigT >

Instantiator for inout operation to generate pixels.

8.301.2 Member Data Documentation

8.301.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxGeneratePix<typename ImgSigT::ArithType> > HxInstGeneratePix::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxGeneratePixInst.c

8.302 HxInstImpBytes Class Template Reference

Instantiator for import operation from bytes.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxImportBytes< typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.302.1 Detailed Description

template<class ImgSigT> class HxInstImpBytes< ImgSigT >

Instantiator for import operation from bytes.

8.302.2 Member Data Documentation

8.302.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxImportBytes<typename ImgSigT::ArithType> > HxInstImpBytes::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxImportBytesInst.c

8.303 HxInstImpPackRgb Class Template Reference

Instantiator for import operation from rgb array.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxImportPackedRgb< typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.303.1 Detailed Description

template<class ImgSigT> class HxInstImpPackRgb< ImgSigT >

Instantiator for import operation from rgb array.

8.303.2 Member Data Documentation

8.303.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxImportPackedRgb<typename ImgSigT::ArithType> > HxInstImpPackRgb::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxImportPackedRgbInst.c

8.304 HxInstImpPpm Class Template Reference

Instantiator for import operation from ppm format.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxImportPpm< typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.304.1 Detailed Description

template<class ImgSigT> class HxInstImpPpm< ImgSigT >

Instantiator for import operation from ppm format.

8.304.2 Member Data Documentation

8.304.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxImportPpm<typename ImgSigT::ArithType> > HxInstImpPpm::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxImportPpmInst.c

8.305 HxInstInOutGetPoints Class Template Reference

Instantiator for inout operation to get points.

Public Attributes

- **HxImgFtorInOut**< ImgSigT, HxInOutGetPoints< typename ImgSigT::ArithType > > **f**
  Instantiate image functor.

8.305.1 Detailed Description

template<class ImgSigT> class HxInstInOutGetPoints< ImgSigT >

Instantiator for inout operation to get points.

8.305.2 Member Data Documentation

8.305.2.1 template<class ImgSigT> HxImgFtorInOut<ImgSigT, HxInOutGetPoints<typename ImgSigT::ArithType> > HxInstInOutGetPoints::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxInOutGetPointsInst.c

8.306 HxInstKerNgb2dNormCorrelation Class Template Reference

Instantiator for kernel based neighbourhood operation with normalized correlation.

Public Attributes

- **HxImgFtorKernelNgb2d**< DstSigT, SrcSigT, KerSigT, **HxKerNgbNormCorrelation**< typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.306.1 Detailed Description

template<**class DstSigT, class SrcSigT, class KerSigT**> **class HxInstKerNgb2dNormCorrelation**< **DstSigT, SrcSigT, KerSigT** >

Instantiator for kernel based neighbourhood operation with normalized correlation.

8.306.2 Member Data Documentation

8.306.2.1 template<class DstSigT, class SrcSigT, class KerSigT> HxImgFtorKernelNgb2d< DstSigT, SrcSigT, KerSigT, HxKerNgbNormCorrelation<typename SrcSigT::ArithType, typename DstSigT::ArithTypeDouble> > HxInstKerNgb2dNormCorrelation::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxKerNgbNormCorrelationInst.c

8.307 HxInstNgb2dMean Class Template Reference

Instantiator for neighbourhood operation with mean.

Public Attributes

- **HxImgFtorNgb2d**< ResSigT, ImgSigT, HxNgbMean< typename ResSigT::ArithType, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.307.1 Detailed Description

template<**class ResSigT, class ImgSigT**> **class HxInstNgb2dMean**< **ResSigT, ImgSigT** >

Instantiator for neighbourhood operation with mean.

8.307.2 Member Data Documentation

8.307.2.1 template<class ResSigT, class ImgSigT> HxImgFtorNgb2d< ResSigT, ImgSigT, HxNgbMean<typename ResSigT::ArithType, typename ImgSigT::ArithType> > HxInstNgb2dMean::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbMeanInst.c

8.308 HxInstNgbIsMaxGradDir2d Class Template Reference

Instantiator for neighbourhood operation with non maximum suppression based on maximum detection.

Public Attributes

- **HxImgFtorNgb2d**< DstSigT, SrcSigT, **HxNgbIsMaxGradDir2d**< typename DstSigT::ArithType, typename SrcSigT::ArithType > > **f**

  Instantiate image functor.

8.308.1 Detailed Description

template<class DstSigT, class SrcSigT> class HxInstNgbIsMaxGradDir2d< DstSigT, SrcSigT >

Instantiator for neighbourhood operation with non maximum suppression based on maximum detection.

8.308.2 Member Data Documentation

8.308.2.1 template<class DstSigT, class SrcSigT> HxImgFtorNgb2d< DstSigT, SrcSigT, HxNgbIsMaxGradDir2d<typename DstSigT::ArithType, typename SrcSigT::ArithType> > HxInstNgbIsMaxGradDir2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbIsMaxGradDir2dInst.c

8.309 HxInstNgbLWshed2d Struct Template Reference

Instantiator for neighbourhood operation with LWshed.

Public Attributes

- **HxImgFtorNgb2d**< ImgSigT, ImgSigT, **HxNgbLWshed2d**< typename ImgSigT::ArithType, typename ImgSigT::ArithType > > **f**

Instantiate image functor.

8.309.1 Detailed Description

template<class ImgSigT> struct HxInstNgbLWshed2d< ImgSigT >

Instantiator for neighbourhood operation with LWshed.

8.309.2 Member Data Documentation

8.309.2.1 template<class ImgSigT> HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbLWshed2d<typename ImgSigT::ArithType, typename ImgSigT::ArithType > > HxInstNgbLWshed2d::f

Instantiate image functor.

The documentation for this struct was generated from the following file:

- HxNgbLWshed2dInst.c

8.310 HxInstNgbNonMaxSuppression2d Class Template Reference

Instantiator for neighbourhood operation with non maximum suppression.

Public Attributes

- **HxImgFtorNgb2d**< ImgSigT, ImgSigT, **HxNgbNonMaxSuppression2d**< typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.310.1 Detailed Description

template<class ImgSigT> class HxInstNgbNonMaxSuppression2d< ImgSigT >

Instantiator for neighbourhood operation with non maximum suppression.

8.310.2 Member Data Documentation

8.310.2.1 template<class ImgSigT> HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbNonMaxSuppression2d<typename ImgSigT::ArithType> > HxInstNgbNonMaxSuppression2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbNonMaxSuppression2dInst.c

8.311 HxInstNgbPercentile2d Class Template Reference

Instantiator for neighbourhood operation with percentile.

Public Attributes

- **HxImgFtorNgb2d**< ImgSigT, ImgSigT, **HxNgbPercentile2d**< typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.311.1 Detailed Description

template<**class ImgSigT**> **class HxInstNgbPercentile2d**< **ImgSigT** >

Instantiator for neighbourhood operation with percentile.

8.311.2 Member Data Documentation

8.311.2.1 template<class ImgSigT> HxImgFtorNgb2d< ImgSigT, ImgSigT, HxNgbPercentile2d<typename ImgSigT::ArithType> > HxInstNgbPercentile2d::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbPercentile2dInst.c

8.312 HxInstRecGenConv2dAddMin Class Template Reference

Instantiator for recursive generalized convolution operation on 2d images with addition and minimum as basic operations.

Public Attributes

- **HxImgFtorRecGenConv2d**< ImgSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoMinAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

8.312.1 Detailed Description

template<**class ImgSigT, class KerSigT**> **class HxInstRecGenConv2dAddMin**< **ImgSigT, KerSigT** >

Instantiator for recursive generalized convolution operation on 2d images with addition and minimum as basic operations.

8.312.2 Member Data Documentation

8.312.2.1 template<class ImgSigT, class KerSigT> HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoMinAssign<typename KerSigT::ArithType, typename KerSigT::ArithType> > HxInstRecGenConv2dAddMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRecGenConvAddMinInst.c

8.313 HxInstRecGenConv2dK1dAddMin Class Template Reference

Instantiator for recursive generalized convolution operation on 2d images with addition and minimum as basic operations.

Public Attributes

- **HxImgFtorRecGenConv2dK1d**< ImgSigT, KerSigT, **HxBpoAdd**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoMinAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

8.313.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstRecGenConv2dK1dAddMin< ImgSigT, KerSigT >

Instantiator for recursive generalized convolution operation on 2d images with addition and minimum as basic operations.

8.313.2 Member Data Documentation

8.313.2.1 template<class ImgSigT, class KerSigT> HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoAdd<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoMinAssign<typename KerSigT::ArithType, typename KerSigT::ArithType> > HxInstRecGenConv2dK1dAddMin::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRecGenConvAddMinInst.c

8.314 HxInstRecGenConv2dK1dMulAdd Class Template Reference

Instantiator for recursive generalized convolution operation on 2d images with multiplication and addition as basic operations.

Public Attributes

- **HxImgFtorRecGenConv2dK1d**< ImgSigT, KerSigT, **HxBpoMul**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoAddAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

8.314.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstRecGenConv2dK1dMulAdd< ImgSigT, KerSigT >

Instantiator for recursive generalized convolution operation on 2d images with multiplication and addition as basic operations.

8.314.2 Member Data Documentation

8.314.2.1 template<class ImgSigT, class KerSigT> HxImgFtorRecGenConv2dK1d< ImgSigT, KerSigT, HxBpoMul<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoAddAssign<typename KerSigT::ArithType, typename KerSigT::ArithType> > HxInstRecGenConv2dK1dMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRecGenConvMulAddInst.c

8.315 HxInstRecGenConv2dMulAdd Class Template Reference

Instantiator for recursive generalized convolution operation on 2d images with multiplication and addition as basic operations.

Public Attributes

- **HxImgFtorRecGenConv2d**< ImgSigT, KerSigT, **HxBpoMul**< typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType >, **HxBpoAddAssign**< typename KerSigT::ArithType, typename KerSigT::ArithType > > **f**

  Instantiate image functor.

8.315.1 Detailed Description

template<class ImgSigT, class KerSigT> class HxInstRecGenConv2dMulAdd< ImgSigT, KerSigT >

Instantiator for recursive generalized convolution operation on 2d images with multiplication and addition as basic operations.

8.315.2 Member Data Documentation

8.315.2.1 template<class ImgSigT, class KerSigT> HxImgFtorRecGenConv2d< ImgSigT, KerSigT, HxBpoMul<typename KerSigT::ArithType, typename KerSigT::ArithType, typename KerSigT::ArithType>, HxBpoAddAssign<typename KerSigT::ArithType, typename KerSigT::ArithType> > HxInstRecGenConv2dMulAdd::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRecGenConvMulAddInst.c

8.316 HxInstRgb2dBinary Class Template Reference

Instantiator for rgb operation on 2d images using binary display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbBinary**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.316.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dBinary< ImgSigT >

Instantiator for rgb operation on 2d images using binary display mapping.

8.316.2 Member Data Documentation

8.316.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbBinary<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dBinary::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbBinaryInst.c

8.317 HxInstRgb2dCMY Class Template Reference

Instantiator for rgb operation on 2d images using CMY display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbCMY**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.317.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb2dCMY**< **ImgSigT** >

Instantiator for rgb operation on 2d images using CMY display mapping.

8.317.2 Member Data Documentation

8.317.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbCMY<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dCMY::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbCMYInst.c

8.318 HxInstRgb2dDirect Class Template Reference

Instantiator for rgb operation on 2d images using direct display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbDirect**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.318.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb2dDirect**< **ImgSigT** >

Instantiator for rgb operation on 2d images using direct display mapping.

8.318.2 Member Data Documentation

8.318.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbDirect<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dDirect::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbDirectInst.c

8.319 HxInstRgb2dDirectNC Class Template Reference

Instantiator for rgb operation on 2d images using direct display mapping without clipping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbDirectNC**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.319.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dDirectNC< ImgSigT >

Instantiator for rgb operation on 2d images using direct display mapping without clipping.

8.319.2 Member Data Documentation

8.319.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbDirectNC<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dDirectNC::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbDirectInst.c

8.320 HxInstRgb2dHSI Class Template Reference

Instantiator for rgb operation on 2d images using HSI display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbHSI**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

Instantiate image functor.

8.320.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dHSI< ImgSigT >

Instantiator for rgb operation on 2d images using HSI display mapping.

8.320.2 Member Data Documentation

8.320.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbHSI<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dHSI::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbHSIInst.c

8.321 HxInstRgb2dLab Class Template Reference

Instantiator for rgb operation on 2d images using Lab display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbLab**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.321.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dLab< ImgSigT >

Instantiator for rgb operation on 2d images using Lab display mapping.

8.321.2 Member Data Documentation

8.321.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbLab<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dLab::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLabInst.c

8.322 HxInstRgb2dLabel Class Template Reference

Instantiator for rgb operation on 2d images using label display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbLabel**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.322.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb2dLabel**< **ImgSigT** >

Instantiator for rgb operation on 2d images using label display mapping.

8.322.2 Member Data Documentation

8.322.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbLabel<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dLabel::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLabelInst.c

8.323 HxInstRgb2dLogMag Class Template Reference

Instantiator for rgb operation on 2d images using log magnitude display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbLogMag**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.323.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb2dLogMag**< **ImgSigT** >

Instantiator for rgb operation on 2d images using log magnitude display mapping.

8.323.2 Member Data Documentation

8.323.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbLogMag<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dLogMag::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLogMagInst.c

8.324 HxInstRgb2dLuv Class Template Reference

Instantiator for rgb operation on 2d images using Luv display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbLuv**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.324.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dLuv< ImgSigT >

Instantiator for rgb operation on 2d images using Luv display mapping.

8.324.2 Member Data Documentation

8.324.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbLuv<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dLuv::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLuvInst.c

8.325 HxInstRgb2dOOO Class Template Reference

Instantiator for rgb operation on 2d images using OOO display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbOOO**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

Instantiate image functor.

8.325.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dOOO< ImgSigT >

Instantiator for rgb operation on 2d images using OOO display mapping.

8.325.2 Member Data Documentation

8.325.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbOOO<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dOOO::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbOOOInst.c

8.326 HxInstRgb2dStretch Class Template Reference

Instantiator for rgb operation on 2d images using stretched display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbStretch**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.326.1 Detailed Description

template<class ImgSigT> class HxInstRgb2dStretch< ImgSigT >

Instantiator for rgb operation on 2d images using stretched display mapping.

8.326.2 Member Data Documentation

8.326.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbStretch<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dStretch::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbStretchInst.c

8.327 HxInstRgb2dXYZ Class Template Reference

Instantiator for rgb operation on 2d images using XYZ display mapping.

Public Attributes

- **HxImgFtorRgb2d**< ImgSigT, **HxRgbXYZ**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.327.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb2dXYZ**< **ImgSigT** >

Instantiator for rgb operation on 2d images using XYZ display mapping.

8.327.2 Member Data Documentation

8.327.2.1 template<class ImgSigT> HxImgFtorRgb2d< ImgSigT, HxRgbXYZ<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb2dXYZ::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbXYZInst.c

8.328 HxInstRgb3dBinary Class Template Reference

Instantiator for rgb operation on 3d images using binary display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbBinary**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.328.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb3dBinary**< **ImgSigT** >

Instantiator for rgb operation on 3d images using binary display mapping.

8.328.2 Member Data Documentation

8.328.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbBinary<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dBinary::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbBinaryInst.c

8.329 HxInstRgb3dCMY Class Template Reference

Instantiator for rgb operation on 3d images using CMY display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbCMY**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.329.1 Detailed Description

template<class ImgSigT> class HxInstRgb3dCMY< ImgSigT >

Instantiator for rgb operation on 3d images using CMY display mapping.

8.329.2 Member Data Documentation

8.329.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbCMY<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dCMY::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbCMYInst.c

8.330 HxInstRgb3dDirect Class Template Reference

Instantiator for rgb operation on 3d images using direct display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbDirect**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

Instantiate image functor.

8.330.1 Detailed Description

template<class ImgSigT> class HxInstRgb3dDirect< ImgSigT >

Instantiator for rgb operation on 3d images using direct display mapping.

8.330.2 Member Data Documentation

8.330.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbDirect<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dDirect::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbDirectInst.c

8.331 HxInstRgb3dHSI Class Template Reference

Instantiator for rgb operation on 3d images using HSI display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbHSI**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.331.1 Detailed Description

template<class ImgSigT> class HxInstRgb3dHSI< ImgSigT >

Instantiator for rgb operation on 3d images using HSI display mapping.

8.331.2 Member Data Documentation

8.331.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbHSI<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dHSI::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbHSIInst.c

8.332 HxInstRgb3dLab Class Template Reference

Instantiator for rgb operation on 3d images using Lab display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbLab**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.332.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb3dLab**< **ImgSigT** >

Instantiator for rgb operation on 3d images using Lab display mapping.

8.332.2 Member Data Documentation

8.332.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbLab<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dLab::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLabInst.c

8.333 HxInstRgb3dLabel Class Template Reference

Instantiator for rgb operation on 3d images using label display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbLabel**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.333.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb3dLabel**< **ImgSigT** >

Instantiator for rgb operation on 3d images using label display mapping.

8.333.2 Member Data Documentation

8.333.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbLabel<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dLabel::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLabelInst.c

8.334 HxInstRgb3dLogMag Class Template Reference

Instantiator for rgb operation on 3d images using log magnitude display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbLogMag**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.334.1 Detailed Description

template<class ImgSigT> class HxInstRgb3dLogMag< ImgSigT >

Instantiator for rgb operation on 3d images using log magnitude display mapping.

8.334.2 Member Data Documentation

8.334.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbLogMag<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dLogMag::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLogMagInst.c

8.335 HxInstRgb3dLuv Class Template Reference

Instantiator for rgb operation on 3d images using Luv display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbLuv**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

Instantiate image functor.

8.335.1 Detailed Description

template<class ImgSigT> class HxInstRgb3dLuv< ImgSigT >

Instantiator for rgb operation on 3d images using Luv display mapping.

8.335.2 Member Data Documentation

8.335.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbLuv<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dLuv::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbLuvInst.c

8.336 HxInstRgb3dOOO Class Template Reference

Instantiator for rgb operation on 3d images using OOO display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbOOO**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.336.1 Detailed Description

template<class ImgSigT> class HxInstRgb3dOOO< ImgSigT >

Instantiator for rgb operation on 3d images using OOO display mapping.

8.336.2 Member Data Documentation

8.336.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbOOO<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dOOO::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbOOOInst.c

8.337 HxInstRgb3dStretch Class Template Reference

Instantiator for rgb operation on 3d images using stretched display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbStretch**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.337.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb3dStretch**< **ImgSigT** >

Instantiator for rgb operation on 3d images using stretched display mapping.

8.337.2 Member Data Documentation

8.337.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbStretch<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dStretch::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbStretchInst.c

8.338 HxInstRgb3dXYZ Class Template Reference

Instantiator for rgb operation on 3d images using XYZ display mapping.

Public Attributes

- **HxImgFtorRgb3d**< ImgSigT, **HxRgbXYZ**< TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble > > **f**

  Instantiate image functor.

8.338.1 Detailed Description

template<**class ImgSigT**> **class HxInstRgb3dXYZ**< **ImgSigT** >

Instantiator for rgb operation on 3d images using XYZ display mapping.

8.338.2 Member Data Documentation

8.338.2.1 template<class ImgSigT> HxImgFtorRgb3d< ImgSigT, HxRgbXYZ<TYPENAME ImgSigT::ArithType, TYPENAME ImgSigT::ArithTypeDouble> > HxInstRgb3dXYZ::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxRgbXYZInst.c

8.339 HxKernel1d Class Template Reference

Standard 1d kernel definition.

`#include <HxKernel.h>`

Public Types

- typedef ArithT **ArithType**

Public Methods

- **HxKernel1d** (KerDataPtrT kerPtr, **HxSizes** size, **HxTagList** &)
- ~**HxKernel1d** ()
- **HxSizes sizes** () const
- const ArithT & **operator()** (int i) const

Static Public Methods

- **HxString className** ()

8.339.1 Detailed Description

template<class KerDataPtrT, class ArithT> class HxKernel1d< KerDataPtrT, ArithT >

Standard 1d kernel definition.

The documentation for this class was generated from the following files:

- **HxKernel.h**
- HxKernel.c

8.340 HxKernel2d Class Template Reference

Standard 2d kernel definition.

`#include <HxKernel.h>`

Public Types

- typedef ArithT **ArithType**

Public Methods

- **HxKernel2d** (KerDataPtrT kerPtr, **HxSizes** size, **HxTagList** &)
- ~**HxKernel2d** ()
- **HxSizes sizes** () const
- ArithT **operator()** (int i, int j) const

Static Public Methods

- **HxString className** ()

8.340.1 Detailed Description

template<class KerDataPtrT, class ArithT> class HxKernel2d< KerDataPtrT, ArithT >

Standard 2d kernel definition.

The documentation for this class was generated from the following files:

- **HxKernel.h**
- HxKernel.c

8.341 HxKernel3d Class Template Reference

Standard 3d kernel definition.

`#include <HxKernel.h>`

Public Types

- typedef ArithT **ArithType**

Public Methods

- **HxKernel3d** (KerDataPtrT kerPtr, **HxSizes** size, **HxTagList** &)
- ~**HxKernel3d** ()
- **HxSizes sizes** () const
- ArithT **operator()** (int i, int j, int k) const

Static Public Methods

- **HxString className** ()

8.341.1 Detailed Description

template<class KerDataPtrT, class ArithT> class HxKernel3d< KerDataPtrT, ArithT >

Standard 3d kernel definition.

The documentation for this class was generated from the following files:

- **HxKernel.h**
- HxKernel.c

8.342 HxKerNgbNormCorrelation Class Template Reference

Neighbourhood functor for normalized (cross) correlation filter.

`#include <HxKerNgbNormCorrelation.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**
  Loop version.
- typedef **HxTag2Phase PhaseCategory**
  2 phases.

Public Methods

- **HxKerNgbNormCorrelation** (**HxTagList** &tags)
  Constructor.
- ~**HxKerNgbNormCorrelation** ()
  Destructor.
- **HxSizes size** ()
  Size of the neighbourhood.
- void **init** (int ix, int iy, ArithT imVal)
  Initialization first phase.
- void **next** (int x, int y, ArithT pixV, ArithT maskV)
  Processing one pixel in first phase.
- void **init2** (int ix, int iy, ArithT imVal)
  Initialization second phase.
- void **next2** (int x, int y, ArithT pixV, ArithT maskV)
  Processing one pixel in second phase.

- ResultT **result** ()
  Produce the result value.

Static Public Methods

- **HxString className** ()
  The name : "normalizedCorrelation".

8.342.1 Detailed Description

template<class ArithT, class ResultT> class HxKerNgbNormCorrelation< 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}

8.342.2 Member Typedef Documentation

8.342.2.1 template<class ArithT, class ResultT> typedef HxTagLoop HxKerNgbNormCorrelation::IteratorCategory

Loop version.

8.342.2.2 template<class ArithT, class ResultT> typedef HxTag2Phase HxKerNgbNormCorrelation::PhaseCategory

2 phases.

8.342.3 Constructor & Destructor Documentation

8.342.3.1 template<class ArithT, class ResultT> HxKerNgbNormCorrelation< ArithT, ResultT >::HxKerNgbNormCorrelation (HxTagList & *tags*)

Constructor.

```
19 {
20     _ngbSize = HxGetTag(tags, "kernelSize", HxSizes(0, 0, 0));
21 }
```

8.342.3.2 template<class ArithT, class ResultT> HxKerNgbNormCorrelation< ArithT, ResultT >::~HxKerNgbNormCorrelation ()

Destructor.

```
25 {
26 }
```

8.342.4 Member Function Documentation

8.342.4.1 template<class ArithT, class ResultT> HxSizes HxKerNgbNormCorrelation< ArithT, ResultT >::size ()

Size of the neighbourhood.

```
39 {
40     return _ngbSize;
41 }
```

8.342.4.2 template<class ArithT, class ResultT> void HxKerNgbNormCorrelation< ArithT, ResultT >::init (int *ix*, int *iy*, ArithT *imVal*) [inline]

Initialization first phase.

```
47 {
48     _num = 0;
49     _fBar = HxScalarDouble(0);
50     _wBar = HxScalarDouble(0);
51 }
```

8.342.4.3 template<class ArithT, class ResultT> void HxKerNgbNormCorrelation< ArithT, ResultT >::next (int *x*, int *y*, ArithT *pixV*, ArithT *maskV*) [inline]

Processing one pixel in first phase.

```
57 {
58     _num += 1;
59     _fBar += pixV;
60     _wBar += maskV;
61 }
```

8.342.4.4 template<class ArithT, class ResultT> void HxKerNgbNormCorrelation< ArithT, ResultT >::init2 (int *ix*, int *iy*, ArithT *imVal*) [inline]

Initialization second phase.

```
66 {
67     _fBar = _fBar / (ResultT) _num;
68     _wBar = _wBar / (ResultT) _num;
69     _sumFW = HxScalarDouble(0);
70     _sumFSqr = HxScalarDouble(0);
71     _sumWSqr = HxScalarDouble(0);
72 }
```

8.342.4.5 template<class ArithT, class ResultT> void HxKerNgbNormCorrelation< ArithT, ResultT >::next2 (int *x*, int *y*, ArithT *pixV*, ArithT *maskV*) [inline]

Processing one pixel in second phase.

```
77 {
78     ResultT p = pixV, m = maskV;
79     _sumFW += (p - _fBar) * (m - _wBar);
80     _sumFSqr += (p - _fBar) * (p - _fBar);
81     _sumWSqr += (m - _wBar) * (m - _wBar);
82 }
```

8.342.4.6 template<class ArithT, class ResultT> ResultT HxKerNgbNormCorrelation< ArithT, ResultT >::result () [inline]

Produce the result value.

```
87 {
88     ResultT tmp = _sumFSqr * _sumWSqr;
89     tmp = tmp.sqrt();
90     if (HxScalarInt(tmp) == HxScalarInt(0))
91         return ResultT(HxScalarInt(0));
92     return _sumFW / tmp;
93 }
```

8.342.4.7 template<class ArithT, class ResultT> HxString HxKerNgbNormCorrelation< ArithT, ResultT >::className () [static]

The name : ”normalizedCorrelation”.

```
31 {
32     static HxString name("normalizedCorrelation");
33     return name;
34 }
```

The documentation for this class was generated from the following files:

- **HxKerNgbNormCorrelation.h**
- HxKerNgbNormCorrelation.c

8.343 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

8.343.1 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. 477) constructor, but that would require a more general version.

8.343.2 Constructor & Destructor Documentation

8.343.2.1 HxLocalInterpol::HxLocalInterpol ()

Default ctor: consistent but useless object.

```
19 {
20     makeDefault();
21 }
```

8.343.2.2 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 }
```

8.343.2.3 HxLocalInterpol::~HxLocalInterpol ()

Destructor.

```
49 {
50 }
```

8.343.3 Member Function Documentation

8.343.3.1 HxPointSetR2 HxLocalInterpol::allP () const [inline]

Get all generated control points.

```
99 {
100     return _points;
101 }
```

8.343.3.2 int HxLocalInterpol::numP () const [inline]

The number of control points.

```
105 {
106     return _points.size();
107 }
```

8.343.3.3 vector< double > HxLocalInterpol::allKnots () const [inline]

Get generated knots vector.

```
111 {
112     return _knots;
113 }
```

8.343.3.4 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

8.344 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**

8.344.1 Detailed Description

Class definition for matrices.

The dimensions are of arbitrary size.

8.344.2 Constructor & Destructor Documentation

8.344.2.1 HxMatrix::HxMatrix () [inline]

Empty matrix.

```
323 {
324     _nr = 0;
325     _nc = 0;
326     _data = 0;
327 }
```

8.344.2.2 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 }
```

8.344.2.3 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 }
```

8.344.2.4 HxMatrix::HxMatrix (int *nRow*, int *nCol*, double * *data*) [inline]

Matrix with given data.

```
352 {
353     _nr = nRow;
354     _nc = nCol;
355     _data = data;
356 }
```

8.344.2.5 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 }
```

8.344.2.6 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 }
```

8.344.2.7 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 }
```

8.344.2.8 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 }
```

8.344.2.9 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 }
```

8.344.2.10 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 }
```

8.344.2.11 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 }
```

8.344.3 Member Function Documentation

8.344.3.1 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 }
```

8.344.3.2 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 }
```

8.344.3.3 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 }
```

8.344.3.4 HxMatrix HxMatrix::rotate2dDeg (double *alpha*) [static]

Rotation in 2D (alpha in deg).

```
261 {
262     return rotate2d(M_PI*alpha/180.0);
263 }
```

8.344.3.5 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 }
```

8.344.3.6 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 }
```

8.344.3.7 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 }
```

8.344.3.8 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 }
```

8.344.3.9 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 }
```

8.344.3.10 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 }
```

8.344.3.11 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 }
```

8.344.3.12 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 }
```

8.344.3.13 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 }
```

8.344.3.14 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 }
```

8.344.3.15 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 }
```

8.344.3.16 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 }
```

8.344.3.17 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 }
```

8.344.3.18 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 }
```

8.344.3.19 int HxMatrix::nRow () const [inline]

Number of rows.

```
378 {
379     return _nr;
380 }
```

8.344.3.20 int HxMatrix::nCol () const [inline]

Number of columns.

```
384 {
385     return _nc;
386 }
```

8.344.3.21 int HxMatrix::nElem () const [inline]

Number of elements.

```
390 {
391     return _nr * _nc;
392 }
```

8.344.3.22 int HxMatrix::valid () const [inline]

Indicates whether the matrix is valid.

```
396 {
397     return ((_nc != 0) && (_nr != 0));
398 }
```

8.344.3.23 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 }
```

8.344.3.24 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 }
```

8.344.3.25 double * HxMatrix::operator[ ] (int *i*) const [inline]

Subscripting, start with 0.

```
429 {
430     return &_data[i*_nc];
431 }
```

8.344.3.26 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 }
```

8.344.3.27 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 }
```

8.344.3.28 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 }
```

8.344.3.29 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 }
```

8.344.3.30 HxMatrix HxMatrix::add (const HxMatrix & *b*) const

Addition.

Equivalent to : a+b

```
684 {
685     return *this+b;
686 }
```

8.344.3.31 HxMatrix HxMatrix::add (const double *val*) const

Addition.

Equivalent to : a+val

```
690 {
691     return *this+val;
692 }
```

8.344.3.32 HxMatrix HxMatrix::sub (const HxMatrix & *b*) const

Subtraction.

Equivalent to : a-b

```
696 {
697     return *this-b;
698 }
```

8.344.3.33 HxMatrix HxMatrix::sub (const double *val*) const

Subtraction.

Equivalent to : a-val

```
702 {
703     return *this-val;
704 }
```

8.344.3.34 HxMatrix HxMatrix::mul (const HxMatrix & *b*) const

Multiplication.

Equivalent to : a∗b

```
708 {
709     return *this*b;
710 }
```

8.344.3.35 HxMatrix HxMatrix::mul (const HxVector & *v*) const

Multiplication.

Equivalent to : a∗v

```
720 {
721     return *this*v;
722 }
```

8.344.3.36 HxMatrix HxMatrix::mul (const double *val*) const

Multiplication.

Equivalent to : a∗val

```
714 {
715     return *this*val;
716 }
```

8.344.3.37 HxMatrix HxMatrix::div (const double *val*) const

Division.

Equivalent to : a/val

```
726 {
727     return *this/val;
728 }
```

8.344.3.38 HxMatrix HxMatrix::sin () const

Apply sin to each element.

```
732 {
733     return map(::sin);
734 }
```

8.344.3.39 HxMatrix HxMatrix::cos () const

Apply cos to each element.

```
738 {
739     return map(::cos);
740 }
```

8.344.3.40 HxMatrix HxMatrix::tan () const

Apply tan to each element.

```
744 {
745     return map(::tan);
746 }
```

8.344.3.41 HxMatrix HxMatrix::sinh () const

Apply sinh to each element.

```
750 {
751     return map(::sinh);
752 }
```

8.344.3.42 HxMatrix HxMatrix::cosh () const

Apply cosh to each element.

```
756 {
757     return map(::cosh);
758 }
```

8.344.3.43 HxMatrix HxMatrix::tanh () const

Apply tanh to each element.

```
762 {
763     return map(::tanh);
764 }
```

8.344.3.44 HxMatrix HxMatrix::exp () const

Apply exp to each element.

```
768 {
769     return map(::exp);
770 }
```

8.344.3.45 HxMatrix HxMatrix::log () const

Apply log to each element.

```
774 {
775     return map(::log);
776 }
```

8.344.3.46 HxMatrix HxMatrix::sqrt () const

Apply sqrt to each element.

```
780 {
781     return map(::sqrt);
782 }
```

8.344.3.47 HxMatrix HxMatrix::abs () const

Apply abs to each element.

```
786 {
787     return map(::fabs);
788 }
```

8.344.3.48 HxMatrix HxMatrix::sgn () const

Apply sgn to each element.

```
794 {
795     return map(::sgn);
796 }
```

8.344.3.49 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 }
```

8.344.4 Friends And Related Function Documentation

8.344.4.1 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 }
```

8.344.4.2 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 }
```

8.344.4.3 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 }
```

8.344.4.4 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 }
```

8.344.4.5 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 }
```

8.344.4.6 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 }
```

8.344.4.7 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 }
```

8.344.4.8 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 }
```

8.344.4.9 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 }
```

8.344.4.10 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 }
```

8.344.4.11 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 }
```

8.344.4.12 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 }
```

8.344.4.13 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 }
```

8.344.4.14 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 }
```

8.344.4.15 int operator!= (const HxMatrix & *a*, const HxMatrix & *b*) [friend]

Not equal.

```
543 {
544     return !(a == b);
545 }
```

8.344.4.16 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 }
```

8.344.4.17 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

8.345 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.

8.345.1 Detailed Description

Class definition of method frame for binary pixel operations.

First queries the **HxImgFtorRuleBase** (p. 840) for an exact match on

resulttype of bpo<src1,src2,bpoName>

If that is not found we look for

argumenttype of bpo<src1,bpoName>

If the argumenttype is not found or it does not match the signature of src2 we try to find

argumenttype of bpo<broadest(src1,src2),bpoName>

If that is also not found, an error is generated. Based on the argumenttype we find the resulttype with

resulttype of bpo<argumenttype,bpoName>

8.345.2 Constructor & Destructor Documentation

8.345.2.1 HxMfBpo::HxMfBpo (HxImageData ∗ *src1*, HxImageData ∗ *src2*, HxString *bpoName*)

Constructor.

```
17     : _src1(src1), _src2(src2), _tmpSrc1(0), _tmpSrc2(0), _preOpIsOk(true)
18 {
19     if (!src1 || !src2)
20     {
21         _preOpIsOk = false;
22         return;
23     }
24
25     HxImageSignature src1Sig(src1->signature());
26     HxImageSignature src2Sig(src2->signature());
27     HxImageSignature broadestSig(src1Sig.broadest(src2Sig));
28     HxImageSignature resultSig;
29
30     HxImgFtorRuleBase::QueryResultType qRes;
31
32     qRes = HxImgFtorRuleBase::instance().getResultType(
33                 src1Sig, "bpo",
34                 src1Sig.toString(), src2Sig.toString(), bpoName);
```

```
35
36      if (qRes)
37      {
38          /*
39           *  Only very specific combinations of argument types allowed.
40           *  Argument types stay what they are.
41           */
42          resultSig = qRes;
43      }
44      else
45      {
46          // Type of result and second argument depends on type of first
47          // argument.
48
49          qRes = HxImgFtorRuleBase::instance().getArgumentType(
50                      src1Sig, "bpo", src1Sig.toString(), bpoName);
51
52          src1Sig = (qRes && src2Sig.isEqual(qRes)) ? src1Sig : broadestSig;
53
54          // If type of second argument does not fit the required type
55          // another attempt is made with the broadest signature.
56          /*  24/01/2001 IMPORTANT DESIGN NOTE:
57           *  One can argue that if a second argument type can be found
58           *  when querying with the original first argument type as key,
59           *  and the given second argument type does not match the query result,
60           *  the second argument should be converted to the query result, but
61           *  only when the query result is broader than the second argument
62           *  type.
63           */
64          // src1Sig = (qRes && src2Sig.broadest(qRes).isEqual(qRes))
65          //              ? src1Sig : broadestSig;
66
67          qRes = HxImgFtorRuleBase::instance().getArgumentType(
68                      src2Sig, "bpo", src1Sig.toString(), bpoName);
69
70          if (!qRes)
71          {
72              HxEnvironment::instance()->errorStream()
73                  << "Cannot apply binary pixel operation " << bpoName << " to "
74                  << "images with type " << src1Sig.toString() << " and "
75                  << src2Sig.toString() << STD_ENDL;
76              HxEnvironment::instance()->flush();
77              _preOpIsOk = false;
78          }
79          else
80          {
81              src2Sig = qRes;
82          }
83
84          resultSig = HxImgFtorRuleBase::instance().getResultType(
85                      src1Sig, "bpo", src1Sig.toString(), bpoName);
86      }
87
88      HxSizes sizes = src1->sizes().sup(src2->sizes());
89
90      if (!src1Sig.isEqual(src1->signature())) {
91          _tmpSrc1 = HxImgDataFactory::instance().makeImage(src1Sig, sizes);
92          _tmpSrc1->set(src1);
93          _src1 = _tmpSrc1;
94      }
95
96      if (!src2Sig.isEqual(src2->signature())) {
97          _tmpSrc2 = HxImgDataFactory::instance().makeImage(src2Sig, sizes);
98          _tmpSrc2->set(src2);
99          _src2 = _tmpSrc2;
```

```
100     }
101
102     _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
103
104 }
```

8.345.2.2 HxMfBpo::~HxMfBpo ()

Destructor.

```
107 {
108     if (_tmpSrc1)
109         delete _tmpSrc1;
110     if (_tmpSrc2)
111         delete _tmpSrc2;
112 }
```

8.345.3 Member Function Documentation

8.345.3.1 HxImageData * HxMfBpo::source1 () const

The first argument image of the frame.

```
116 {
117     return _src1;
118 }
```

8.345.3.2 HxImageData * HxMfBpo::source2 () const

The second argument image of the frame.

```
122 {
123     return _src2;
124 }
```

8.345.3.3 HxImageData * HxMfBpo::result () const

The result image of the frame.

```
128 {
129     return _result;
130 }
```

8.345.3.4 bool HxMfBpo::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
72 {
73     return _preOpIsOk;
74 }
```

The documentation for this class was generated from the following files:

- **HxMfBpo.h**
- HxMfBpo.c

8.346 HxMfDiy Class Reference

Class definition of method frame for do it yourself operations.

`#include <HxMfDiy.h>`

Public Methods

- **HxMfDiy** (**HxImageData** *srcImg, **HxString** diyName, **HxSizes** resultSize)
  Constructor.
- ~**HxMfDiy** ()
  Destructor.
- **HxImageData** * **source** () const
  The argument image of the frame.
- **HxImageData** * **result** () const
  The result image of the frame.

8.346.1 Detailed Description

Class definition of method frame for do it yourself operations.

A result image will be allocated with the given size. The type of the result image is obtained from the **HxImgFtorRuleBase** (p. 840) via

resulttype of diy<source,diyName>

8.346.2 Constructor & Destructor Documentation

8.346.2.1 HxMfDiy::HxMfDiy (HxImageData * *srcImg*, HxString *diyName*, HxSizes *resultSize*)

Constructor.

```
19     : _source(srcImg), _result(0)
20 {
21     HxImageSignature srcSig(_source->signature());
22
23     HxImageSignature resultSig
24         = HxImgFtorRuleBase::instance().getResultType(
25                 srcSig, "diy", srcSig.toString(), diyName);
26
27     _result = HxImgDataFactory::instance().makeImage(resultSig, resultSize);
28 }
```

8.346.2.2 HxMfDiy::~HxMfDiy ()

Destructor.

```
31 {
32 }
```

8.346.3 Member Function Documentation

8.346.3.1 HxImageData * HxMfDiy::source () const

The argument image of the frame.

```
36 {
37     return _source;
38 }
```

8.346.3.2 HxImageData * HxMfDiy::result () const

The result image of the frame.

```
42 {
43     return _result;
44 }
```

The documentation for this class was generated from the following files:

- **HxMfDiy.h**
- HxMfDiy.c

8.347 HxMfExportExtra Class Reference

Class definition of a method frame for export operations using an extra image.

`#include <HxMfExportExtra.h>`

Public Methods

- **HxMfExportExtra** (**HxImageData** *source, **HxImageData** *extra, **HxString** exName, **HxTagList** &tags)

  Constructor.

- **~HxMfExportExtra** ()

  Destructor.

- **HxImageData** * **source** () const

  The source image of the frame.

- **HxImageData** * **extra** () const

The extra image of the frame.

- bool **preOpIsOk** () const

  Indicates whether initialization was OK.

8.347.1 Detailed Description

Class definition of a method frame for export operations using an extra image.

The required type for the extra image is obtained via

extratype of exportExtra<source,exportName>

8.347.2 Constructor & Destructor Documentation

8.347.2.1 HxMfExportExtra::HxMfExportExtra (HxImageData * *srcImg*, HxImageData * *extra*, HxString *exName*, HxTagList & *tags*)

Constructor.

```
20     : _source(srcImg), _extra(extra), _tmpExtra(0), _preOpIsOk(true)
21 {
22     if (!_source || !_extra) {
23         _preOpIsOk = false;
24         return;
25     }
26
27     HxImageSignature srcSig(_source->signature());
28
29     HxImageSignature extraSig
30         = HxImgFtorRuleBase::instance().getExtraType(
31             _extra->signature(), "exportExtra", srcSig.toString(), exName);
32
33     if (extraSig != _extra->signature()) {
34         _tmpExtra = HxImgDataFactory::instance().makeImage(
35             extraSig, _extra->sizes());
36         _tmpExtra->setPartImage(_extra);
37         _extra = _tmpExtra;
38     }
39 }
```

8.347.2.2 HxMfExportExtra::~HxMfExportExtra ()

Destructor.

```
42 {
43     if (_tmpExtra)
44         delete _tmpExtra;
45 }
```

8.347.3 Member Function Documentation

8.347.3.1 HxImageData * HxMfExportExtra::source () const

The source image of the frame.

```
49 {
50     return _source;
51 }
```

8.347.3.2 HxImageData * HxMfExportExtra::extra () const

The extra image of the frame.

```
55 {
56     return _extra;
57 }
```

8.347.3.3 bool HxMfExportExtra::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
54 {
55     return _preOpIsOk;
56 }
```

The documentation for this class was generated from the following files:

- **HxMfExportExtra.h**
- HxMfExportExtra.c

8.348 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)

  Constructor.

- **HxMfGenConv** (**HxImageData** *source, **HxImageData** *kernel, **HxImageData** *kernel2, **HxImageRep::ResultPrecision** resPrec, bool is1dConv=false)

  Constructor.

- **HxMfGenConv** (**HxImageData** *source, **HxImageData** *kernel, **HxImageData** *kernel2, **HxImageData** *kernel3, **HxImageRep::ResultPrecision** resPrec, bool is1dConv=false)

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** ∗ **kernel3** () const

  The third kernel image of the frame.

- **HxImageData** ∗ **result** () const

  The result image of the frame.

- bool **preOpIsOk** () const

  Indicates whether initialization was OK.

8.348.1 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.

If is1dConv is false, 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.

Note that the actual convolution will be performed in the precision of the kernel type after conversion. After that, the result is written to the output image

8.348.2 Constructor & Destructor Documentation

8.348.2.1 HxMfGenConv::HxMfGenConv (HxImageData * *source*, HxImageData * *kernel*, HxImageRep::ResultPrecision *resPrec*, bool *is1dConv* = false)

Constructor.

```
21     :   _preOpIsOk(true), _source(source),
22         _kernel(kernel), _tmpKernel(0), _kernel2(0), _tmpKernel2(0),
23         _kernel3(0), _tmpKernel3(0)
24 {
25     initMethodFrame(resPrec, is1dConv);
26 }
```

8.348.2.2 HxMfGenConv::HxMfGenConv (HxImageData * *source*, HxImageData * *kernel*, HxImageData * *kernel2*, HxImageRep::ResultPrecision *resPrec*, bool *is1dConv* = false)

Constructor.

```
31     :   _preOpIsOk(true), _source(source),
32         _kernel(kernel), _tmpKernel(0), _kernel2(kernel2), _tmpKernel2(0),
33         _kernel3(0), _tmpKernel3(0)
34 {
35     initMethodFrame(resPrec, is1dConv);
36 }
```

8.348.2.3 HxMfGenConv::HxMfGenConv (HxImageData * *source*, HxImageData * *kernel*, HxImageData * *kernel2*, HxImageData * *kernel3*, HxImageRep::ResultPrecision *resPrec*, bool *is1dConv* = false)

Constructor.

```
42     :   _preOpIsOk(true), _source(source),
43         _kernel(kernel), _tmpKernel(0), _kernel2(kernel2), _tmpKernel2(0),
44         _kernel3(kernel3), _tmpKernel3(0)
45 {
46     initMethodFrame(resPrec, is1dConv);
47 }
```

8.348.2.4 HxMfGenConv::~HxMfGenConv ()

Destructor.

```
50 {
51     if (_tmpKernel)
52         delete _tmpKernel;
53     if (_tmpKernel2)
54         delete _tmpKernel2;
55     if (_tmpKernel3)
56         delete _tmpKernel3;
57 }
```

8.348.3 Member Function Documentation

8.348.3.1 HxImageData * HxMfGenConv::source () const

The source image of the frame.

```
61 {
62     return _source;
63 }
```

8.348.3.2 HxImageData * HxMfGenConv::kernel () const

The kernel image of the frame.

```
67 {
68     return _kernel;
69 }
```

8.348.3.3 HxImageData * HxMfGenConv::kernel2 () const

The second kernel image of the frame.

```
73 {
74     return _kernel2;
75 }
```

8.348.3.4 HxImageData * HxMfGenConv::kernel3 () const

The third kernel image of the frame.

```
79 {
80     return _kernel3;
81 }
```

8.348.3.5 HxImageData * HxMfGenConv::result () const

The result image of the frame.

```
85 {
86     return _result;
87 }
```

8.348.3.6 bool HxMfGenConv::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
120 {
121     return _preOpIsOk;
122 }
```

The documentation for this class was generated from the following files:

- **HxMfGenConv.h**
- HxMfGenConv.c

8.349 HxMfIdentity Class Reference

Class definition of method frame for identity.

`#include <HxMfIdentity.h>`

Public Methods

- **HxMfIdentity** (**HxImageData** *src, int pixDim=0)
  Constructor.
- ~**HxMfIdentity** ()
  Destructor.
- **HxImageData** * **source** () const
  The source image of the frame.
- **HxImageData** * **result** () const
  The result image of the frame.

8.349.1 Detailed Description

Class definition of method frame for identity.

The method frame just copies the image data to ensure the value paradigm. So, **result**() (p. 1018) will point to a copy of src (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.

8.349.2 Constructor & Destructor Documentation

8.349.2.1 HxMfIdentity::HxMfIdentity (HxImageData * *src*, int *pixDim* = 0)

Constructor.

```
16                                                      : _source(src)
17 {
18     if (!_source)
19         return;
20
21     HxImageSignature srcSig(_source->signature());
22     if (pixDim != 0)
23         srcSig.setPixelDimensionality(pixDim);
24     HxSizes sizes = _source->sizes();
25
26     _result = HxImgDataFactory::instance().makeImage(srcSig, sizes);
```

```
27     if (_result)
28         _result->set(_source);
29 }
```

8.349.2.2 HxMfIdentity::~HxMfIdentity ()

Destructor.

```
32 {
33 }
```

8.349.3 Member Function Documentation

8.349.3.1 HxImageData * HxMfIdentity::source () const

The source image of the frame.

```
37 {
38     return _source;
39 }
```

8.349.3.2 HxImageData * HxMfIdentity::result () const

The result image of the frame.

```
43 {
44     return _result;
45 }
```

The documentation for this class was generated from the following files:

- **HxMfIdentity.h**
- HxMfIdentity.c

8.350 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.

8.350.1 Detailed Description

Class definition of a method frame for neighbourhood operations using a kernel.

A result image will be allocated with the same size as the source image. The type of the result image is obtained from the **HxImgFtorRuleBase** (p. 840) via

resulttype of kernelNgb<source,ngbName>

The required type for the kernel is obtained via

kerneltype of kernelNgb<source,ngbName>

8.350.2 Constructor & Destructor Documentation

8.350.2.1 HxMfKernelNgb::HxMfKernelNgb (HxImageData ∗ *srcImg*, HxImageData ∗ *kernel*, HxString *ngbName*, HxTagList & *tags*)

Constructor.

```
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 }
```

8.350.2.2 HxMfKernelNgb::~HxMfKernelNgb ()

Destructor.

```
53 {
54     if (_tmpKernel)
55         delete _tmpKernel;
56 }
```

8.350.3 Member Function Documentation

8.350.3.1 HxImageData * HxMfKernelNgb::source () const

The source image of the frame.

```
60 {
61     return _source;
62 }
```

8.350.3.2 HxImageData * HxMfKernelNgb::kernel () const

The kernel image of the frame.

```
66 {
67     return _kernel;
68 }
```

8.350.3.3 HxImageData * HxMfKernelNgb::result () const

The result image of the frame.

```
72 {
73     return _result;
74 }
```

8.350.3.4 bool HxMfKernelNgb::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
64 {
65     return _preOpIsOk;
66 }
```

The documentation for this class was generated from the following files:

- **HxMfKernelNgb.h**
- HxMfKernelNgb.c

8.351 HxMfMNpo Class Reference

Class definition of method frame for M output N input pixel pixel operations.

`#include <HxMfMNpo.h>`

Public Methods

- **HxMfMNpo** (**HxImageData** ∗∗srcs, int srcCnt, **HxString** mnpoName)
  Constructor.
- ∼**HxMfMNpo** ()
  Destructor.
- **HxImageData** ∗∗ **results** () const
  The result images of the frame.
- **HxImageData** ∗ **results** (int n) const
  The n-th result image of the frame.
- int **resultCnt** () const
  The number of result 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.

8.351.1 Detailed Description

Class definition of method frame for M output N input pixel pixel operations.

First we determine the broadest signature and supremum of sizes of all source images. The sources images are prepared to match that signature and sizes.

The number of result images is determined by calling HxImageData::probeMNpo. Result images will be allocated with the same size as the source images. The type of the result images is obtained from the **HxImgFtorRuleBase** (p. 840) via

resulttype of mnpo<broadest,mnpoName>

8.351.2 Constructor & Destructor Documentation

8.351.2.1 HxMfMNpo::HxMfMNpo (HxImageData ** *srcs*, int *srcCnt*, HxString *mnpoName*)

Constructor.

```
18     : _preOpIsOk(true)
19 {
20     if (srcCnt <= 0) {
21         _preOpIsOk = false;
22         return;
23     }
24
25     _srcCnt = srcCnt;
26     _src = new HxImageData * [srcCnt];
27     _tmp = new HxImageData * [srcCnt];
28
29     int i;
30     for (i = 0; i < srcCnt; i++) {
31         if (!srcs[i]) {
32             _preOpIsOk = false;
33             return;
34         }
35         _src[i] = srcs[i];
36         _tmp[i] = 0;
37     }
38
39     /* find broadest signature and supremum of sizes */
40     HxImageSignature broadestSig(srcs[0]->signature());
41     HxSizes sizes = srcs[0]->sizes();
42     for (i = 1; i < srcCnt; i++) {
43         HxImageSignature srcnSig(srcs[i]->signature());
44         broadestSig = broadestSig.broadest(srcnSig);
45         sizes = sizes.sup(srcs[i]->sizes());
46     }
47
48     HxImageSignature resultSig
49         = HxImgFtorRuleBase::instance().getResultType(
50             broadestSig, "mnpo", broadestSig.toString(), mnpoName);
51
52     for (i = 0; i < srcCnt; i++) {
53         if (!broadestSig.isEqual(srcs[i]->signature())) {
54             _tmp[i] = HxImgDataFactory::instance().makeImage(broadestSig, sizes);
55             _tmp[i]->set(srcs[i]);
56             _src[i] = _tmp[i];
57         }
58     }
59
61     HxTagList tags;
62     HxAddTag(tags, "sourceCnt", _srcCnt);
63     if (!HxImageData::probeMNpo(resultSig, broadestSig, mnpoName, tags)) {
64         _preOpIsOk = false;
65         return;
66     }
67     _resCnt = HxGetTag(tags, "resultCnt", 1);
68
69     _result = new HxImageData * [_resCnt];
70     for (i = 0; i < _resCnt; i++)
71         _result[i] = HxImgDataFactory::instance().makeImage(resultSig, sizes);
72 }
```

8.351.2.2 HxMfMNpo::~HxMfMNpo ()

Destructor.

```
75 {
76     for (int i = 0; i < _srcCnt; i++) {
77         if (_tmp[i])
78             delete _tmp[i];
79     }
80     delete [] _result;
81     delete [] _tmp;
82     delete [] _src;
83 }
```

8.351.3 Member Function Documentation

8.351.3.1 HxImageData ** HxMfMNpo::results () const

The result images of the frame.

```
99 {
100     return _result;
101 }
```

8.351.3.2 HxImageData * HxMfMNpo::results (int *n*) const [inline]

The n-th result image of the frame.

```
75 {
76     return ((n>=0) && (n < _resCnt)) ? _result[n] : 0;
77 }
```

8.351.3.3 int HxMfMNpo::resultCnt () const

The number of result images of the frame.

```
105 {
106     return _resCnt;
107 }
```

8.351.3.4 HxImageData ** HxMfMNpo::sources () const

The source images of the frame.

```
87 {
88     return _src;
89 }
```

8.351.3.5 int HxMfMNpo::sourceCnt () const

The number of source images of the frame.

```
93 {
94     return _srcCnt;
95 }
```

8.351.3.6 bool HxMfMNpo::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
69 {
70     return _preOpIsOk;
71 }
```

The documentation for this class was generated from the following files:

- **HxMfMNpo.h**
- HxMfMNpo.c

8.352 HxMfMpo Class Reference

Class definition of method frame for multi 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.

8.352.1 Detailed Description

Class definition of method frame for multi pixel operations.

First we determine the broadest signature and supremum of sizes of all source images. The sources images are prepared to match that signature and sizes.

A result image will be allocated with the same size as the source images. The type of the result image is obtained from the **HxImgFtorRuleBase** (p. 840) via

resulttype of mpo<broadest,mpoName>

8.352.2 Constructor & Destructor Documentation

8.352.2.1 HxMfMpo::HxMfMpo (HxImageData ** *srcs*, int *nSrcs*, HxString *mpoName*)

Constructor.

```
18 {
19     if (!nSrcs)
20         return;
21
22     _nSrcs = nSrcs;
23     _src = new HxImageData * [nSrcs];
24     _tmp = new HxImageData * [nSrcs];
25
26     int i;
27     for (i = 0; i < nSrcs; i++) {
28         if (!srcs[i])
29             return;
30         _src[i] = srcs[i];
31         _tmp[i] = 0;
32     }
33
34     /* find broadest signature and supremum of sizes */
35     HxImageSignature broadestSig(srcs[0]->signature());
36     HxSizes sizes = srcs[0]->sizes();
37     for (i = 1; i < nSrcs; i++) {
38         HxImageSignature srcnSig(srcs[i]->signature());
39         broadestSig = broadestSig.broadest(srcnSig);
40         sizes = sizes.sup(srcs[i]->sizes());
41     }
42
43     HxImageSignature resultSig
44         = HxImgFtorRuleBase::instance().getResultType(
45             broadestSig, "mpo", broadestSig.toString(), mpoName);
46
47     for (i = 0; i < nSrcs; i++) {
48         if (!broadestSig.isEqual(srcs[i]->signature())) {
49             _tmp[i] = HxImgDataFactory::instance().makeImage(broadestSig, sizes);
50             _tmp[i]->set(srcs[i]);
51             _src[i] = _tmp[i];
52         }
53     }
54
55     _result = HxImgDataFactory::instance().makeImage(resultSig, sizes);
56 }
```

8.352.2.2 HxMfMpo::~HxMfMpo ()

Destructor.

```
59 {
60     for (int i = 0; i < _nSrcs; i++) {
61         if (_tmp[i])
62             delete _tmp[i];
63     }
64     delete [] _tmp;
65     delete [] _src;
66 }
```

8.352.3 Member Function Documentation

8.352.3.1 HxImageData ** HxMfMpo::sources () const

The source images of the frame.

```
70 {
71     return _src;
72 }
```

8.352.3.2 int HxMfMpo::nSources () const

The number of source images of the frame.

```
76 {
77     return _nSrcs;
78 }
```

8.352.3.3 HxImageData * HxMfMpo::result () const

The result image of the frame.

```
82 {
83     return _result;
84 }
```

The documentation for this class was generated from the following files:

- **HxMfMpo.h**
- HxMfMpo.c

8.353 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** (**HxImageData** ∗srcImg, **HxImageData** ∗extraIm, **HxString** ngbName, **HxTagList** &tags)
  Constructor.
- **HxMfNgb** (**HxImageData** ∗srcImg, **HxImageData** ∗extraIm, **HxImageData** ∗extraIm2, **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.
- **HxImageData** ∗ **extra** () const
  The extra image of the frame.
- **HxImageData** ∗ **extra2** () const
  The second extra image of the frame.
- bool **preOpIsOk** () const
  Indicates whether initialization was OK.

8.353.1 Detailed Description

Class definition of method frame for neighbourhood operations.

A result image will be allocated with the same size as the source image. The type of the result image is obtained from the **HxImgFtorRuleBase** (p. 840) via

resulttype of ngb<source,ngbName>

The required type for the extra images, if present, are obtained via

extratype of ngb<source,ngbName> and extratype2 of ngb<source,ngbName>

8.353.2 Constructor & Destructor Documentation

8.353.2.1 HxMfNgb::HxMfNgb (HxImageData ∗ *srcImg*, HxString *ngbName*, HxTagList & *tags*)

Constructor.

```
20     : _source(srcImg), _extra(0), _extra2(0), _result(0), _preOpIsOk(true),
21       _tmpExtra(0), _tmpExtra2(0)
22 {
23     init(ngbName, tags);
24 }
```

8.353.2.2 HxMfNgb::HxMfNgb (HxImageData * *srcImg*, HxImageData * *extraIm*, HxString *ngbName*, HxTagList & *tags*)

Constructor.

```
28     : _source(srcImg), _extra(extraIm), _extra2(0), _result(0), _preOpIsOk(true),
29       _tmpExtra(0), _tmpExtra2(0)
30 {
31     init(ngbName, tags);
32 }
```

8.353.2.3 HxMfNgb::HxMfNgb (HxImageData * *srcImg*, HxImageData * *extraIm*, HxImageData * *extraIm2*, HxString *ngbName*, HxTagList & *tags*)

Constructor.

```
36     : _source(srcImg), _extra(extraIm), _extra2(extraIm2), _result(0),
37       _preOpIsOk(true), _tmpExtra(0), _tmpExtra2(0)
38 {
39     init(ngbName, tags);
40 }
```

8.353.2.4 HxMfNgb::~HxMfNgb ()

Destructor.

```
43 {
44     if (_tmpExtra)
45         delete _tmpExtra;
46     if (_tmpExtra2)
47         delete _tmpExtra2;
48 }
```

8.353.3 Member Function Documentation

8.353.3.1 HxImageData * HxMfNgb::source () const

The argument image of the frame.

```
52 {
53     return _source;
54 }
```

8.353.3.2 HxImageData * HxMfNgb::result () const

The result image of the frame.

```
58 {
59     return _result;
60 }
```

8.353.3.3 HxImageData * HxMfNgb::extra () const

The extra image of the frame.

```
64 {
65     return _extra;
66 }
```

8.353.3.4 HxImageData * HxMfNgb::extra2 () const

The second extra image of the frame.

```
70 {
71     return _extra2;
72 }
```

8.353.3.5 bool HxMfNgb::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
78 {
79     return _preOpIsOk;
80 }
```

The documentation for this class was generated from the following files:

- **HxMfNgb.h**
- HxMfNgb.c

8.354 HxMfQueueBased Class Reference

Class definition of a method frame for queue based operations.

`#include <HxMfQueueBased.h>`

Public Methods

- **HxMfQueueBased** (**HxImageData** *source, **HxImageData** *kernel, **HxString** ngbName, **HxTagList** &tags)

  Constructor.

- ~**HxMfQueueBased** ()

  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.

8.354.1 Detailed Description

Class definition of a method frame for queue based operations.

8.354.2 Constructor & Destructor Documentation

8.354.2.1 HxMfQueueBased::HxMfQueueBased (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 #ifdef _DEBUG
28         std::cout << "false at line " << __LINE__ << std::endl;
29 #endif //#ifdef _DEBUG
30         return;
31     }
32
33 #ifdef _DEBUG
34     std::cout << "srcImg= " << srcImg->signature().toString() << " " << srcImg->sizes() << std::endl;
35     std::cout << "kernel= " << kernel->signature().toString() << " " << kernel->sizes() << std::endl;
36     std::cout << "ngbName= " << ngbName << std::endl;
37     std::cout << "tags=" << tags << std::endl;
38 #endif //#ifdef _DEBUG
39     HxImageSignature srcSig(_source->signature());
40
41     HxImageSignature resultSig
42         = HxImgFtorRuleBase::instance().getResultType(
43             srcSig, "QueueBased", srcSig.toString(), ngbName);
```

```
44
45     HxImageSignature kernelSig
46         = HxImgFtorRuleBase::instance().getKernelType(
47             _kernel->signature(), "QueueBased", srcSig.toString(), ngbName);
48
49     _result = HxImgDataFactory::instance().makeImage(resultSig, srcImg->sizes());
50
51     if (kernelSig != _kernel->signature())
52     {
53         _tmpKernel = HxImgDataFactory::instance().makeImage(
54             kernelSig, _kernel->sizes());
55         _tmpKernel->setPartImage(_kernel);
56         _tmpKernel->weight(_kernel->weight().x());
57         _kernel = _tmpKernel;
58     }
59 #ifdef _DEBUG
60     std::cout << "_source= " << _source->signature().toString() << " " << _source->sizes() << std::endl;
61     std::cout << "_kernel= " << _kernel->signature().toString() << " " << _kernel->sizes() << std::endl;
62     std::cout << "_result= " << _result->signature().toString() << " " << _result->sizes() << std::endl;
63 #endif //#ifdef _DEBUG
64 }
```

8.354.2.2 HxMfQueueBased::~HxMfQueueBased ()

Destructor.

```
67 {
68     if (_tmpKernel)
69         delete _tmpKernel;
70 }
```

8.354.3 Member Function Documentation

8.354.3.1 HxImageData * HxMfQueueBased::source () const

The source image of the frame.

```
74 {
75     return _source;
76 }
```

8.354.3.2 HxImageData * HxMfQueueBased::kernel () const

The kernel image of the frame.

```
80 {
81     return _kernel;
82 }
```

8.354.3.3 HxImageData * HxMfQueueBased::result () const

The result image of the frame.

```
86 {
87     return _result;
88 }
```

8.354.3.4 bool HxMfQueueBased::preOpIsOk () const [inline]

Indicates whether initialization was OK.

```
59 {
60     return _preOpIsOk;
61 }
```

The documentation for this class was generated from the following files:

- **HxMfQueueBased.h**
- HxMfQueueBased.c

8.355 HxMfResize Class Reference

Class definition of method frame for resizing of image.

`#include <HxMfResize.h>`

Public Methods

- **HxMfResize** (**HxImageData** *src, **HxSizes** newSize, **HxImageData** *argImg=0)
  Constructor.
- ~**HxMfResize** ()
  Destructor.
- **HxImageData** * **source** () const
  The source image of the frame.
- **HxImageData** * **argument** () const
  The argument image of the frame.
- **HxImageData** * **result** () const
  The result image of the frame.

8.355.1 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.

result() (p. 1033) will point to an empty image with the signature of src and the specified sizes. In case the signature of argImg is not equal to the signature of src a temporary image with that signature will be created for **argument**() (p. 1033) with the same pixel values as argImg.

8.355.2 Constructor & Destructor Documentation

8.355.2.1 HxMfResize::HxMfResize (HxImageData * *src*, HxSizes *newSize*, HxImageData * *argImg* = 0)

Constructor.

```
17     : _source(src), _argument(argImg), _tmpArg(0)
18 {
19     if (!_source)
20         return;
21
22     HxImageSignature srcSig(_source->signature());
23
24     if (_argument) {
25         HxImageSignature argSig(_argument->signature());
26         if (!argSig.isEqual(srcSig)) {
27             _tmpArg = HxImgDataFactory::instance().makeImage(srcSig, newSize);
28             _tmpArg->set(_argument);
29             _argument = _tmpArg;
30         }
31     }
32
33     _result = HxImgDataFactory::instance().makeImage(srcSig, newSize);
34 //    if (_result)
35 //        _result->set(_source);
36 }
```

8.355.2.2 HxMfResize::~HxMfResize ()

Destructor.

```
39 {
40     if (_tmpArg)
41         delete _tmpArg;
42 }
```

8.355.3 Member Function Documentation

8.355.3.1 HxImageData * HxMfResize::source () const

The source image of the frame.

```
46 {
47     return _source;
48 }
```

8.355.3.2 HxImageData * HxMfResize::argument () const

The argument image of the frame.

```
52 {
53     return _argument;
54 }
```

8.355.3.3 HxImageData * HxMfResize::result () const

The result image of the frame.

```
58 {
59     return _result;
60 }
```

The documentation for this class was generated from the following files:

- **HxMfResize.h**
- HxMfResize.c

8.356 HxMfUpo Class Reference

Class definition of method frame for unary pixel operations.

`#include <HxMfUpo.h>`

Public Methods

- **HxMfUpo** (**HxImageData** *src, **HxString** upoName)
  Constructor.
- ~**HxMfUpo** ()
  Destructor.
- **HxImageData** * **source** () const
  The source image of the frame.
- **HxImageData** * **result** () const
  The result image of the frame.

8.356.1 Detailed Description

Class definition of method frame for unary pixel operations.

A result image will be allocated with the same size as the source image. The type of the result image is obtained from the **HxImgFtorRuleBase** (p. 840) via

resulttype of upo<src,upoName>

8.356.2 Constructor & Destructor Documentation

8.356.2.1 HxMfUpo::HxMfUpo (HxImageData * *src*, HxString *upoName*)

Constructor.

```
18     : _source(src)
19 {
20     if (!_source)
21         return;
22
23     HxImageSignature srcSig(_source->signature());
24
25     HxImgFtorRuleBase::QueryResultType resultSig
26         = HxImgFtorRuleBase::instance().getResultType(
27             srcSig, "upo", srcSig.toString(), upoName);
28
29     if (int(resultSig)) {
30         HxSizes sizes = _source->sizes();
31
32         _result = HxImgDataFactory::instance().makeImage(
33             HxImageSignature(resultSig), sizes);
34     }
35     else
36         _result = 0;
37 }
```

8.356.2.2 HxMfUpo::~HxMfUpo ()

Destructor.

```
40 {
41 }
```

8.356.3 Member Function Documentation

8.356.3.1 HxImageData * HxMfUpo::source () const

The source image of the frame.

```
45 {
46     return _source;
47 }
```

8.356.3.2 HxImageData * HxMfUpo::result () const

The result image of the frame.

```
51 {
52     return _result;
53 }
```

The documentation for this class was generated from the following files:

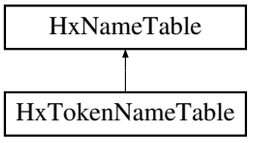
- **HxMfUpo.h**
- HxMfUpo.c

8.357 HxNameTable Class Reference

Class definition of a name table.

`#include <HxNameTable.h>`

Inheritance diagram for HxNameTable::



Public Types

- typedef size_t **sizeType**
  The type of the Id's.

Public Methods

- **HxNameTable** ()
  Constructor.
- virtual ~**HxNameTable** ()
  Destructor.
- void **insert** (**HxString** name, **sizeType** id)
  Insert name,id combination.
- **sizeType getId** (**HxString** name)
  Get the id for this name.
- **HxString getName** (**sizeType** id) const
  Get the name for this id.
- std::vector< **HxString** > **getNames** () const
  Get all names in the table.
- std::ostream & **put** (std::ostream &) const
  Put the table on the stream.

8.357.1 Detailed Description

Class definition of a name table.

A name table is used to maintain a list of name,id combinations. The id's are unique.

8.357.2 Member Typedef Documentation

8.357.2.1 typedef size_t HxNameTable::sizeType

The type of the Id's.

8.357.3 Constructor & Destructor Documentation

8.357.3.1 HxNameTable::HxNameTable ()

Constructor.

```
15 {
16 }
```

8.357.3.2 HxNameTable::~HxNameTable () [virtual]

Destructor.

```
19 {
20 }
```

8.357.4 Member Function Documentation

8.357.4.1 void HxNameTable::insert (HxString *name*, sizeType *i*)

Insert name,id combination.

```
24 {
25     Map::iterator ptr;
26     HxString oldName("");
27 
28     if (i >= _index.size()) {
29         _index.insert(_index.end(), i-_index.size()+1, HxString(""));
30     }
31 
32     oldName = _index[i];
33     if ((!oldName.empty()) && (oldName != name)) {
34         ptr = _map.find(oldName);
35         if (ptr != _map.end()) {
36             _map.erase(ptr);
37         }
38     }
39 
40     if ((ptr = _map.find(name)) != _map.end()) {
41         _index[(*ptr).second] = "";
42     }
43 
44     _map[name] = i;
45     _index[i] = name;
46 }
```

8.357.4.2 HxNameTable::sizeType HxNameTable::getId (HxString *name*)

Get the id for this name.

```
50 {
51     Map::iterator ptr = _map.find(name);
52     if (ptr != _map.end()) {
53         return (*ptr).second;
54     } else {
55         _index.push_back(name);
56         _map.insert(Entry(name, _index.size()-1));
57         return _index.size()-1;
58     }
59 }
```

8.357.4.3 HxString HxNameTable::getName (sizeType *id*) const

Get the name for this id.

```
63 {
64     return id < _index.size() ? _index[id] : HxString("");
65 }
```

8.357.4.4 std::vector< HxString > HxNameTable::getNames () const

Get all names in the table.

```
69 {
70     return _index;
71 }
```

8.357.4.5 std::ostream & HxNameTable::put (std::ostream & *os*) const

Put the table on the stream.

```
75 {
76     Map::const_iterator     mapPtr;
77     Index::const_iterator   idxPtr;
78     sizeType                i;
79
80     os << "MAP" << STD_ENDL;
81     for (mapPtr = _map.begin(); mapPtr != _map.end(); ++mapPtr) {
82         os << (*mapPtr).first << " " << (*mapPtr).second << STD_ENDL;
83     }
84     os << "INDEX" << STD_ENDL;
85     for (i=0, idxPtr = _index.begin(); idxPtr != _index.end(); ++i, ++idxPtr) {
86         os << i << " " << *idxPtr << STD_ENDL;
87     }
88
89     return os;
90 }
```

The documentation for this class was generated from the following files:

- **HxNameTable.h**
- HxNameTable.c

8.358 HxNgbBernsen Class Template Reference

Neighbourhood functor for bernsen thresholding.

`#include <HxNgbBernsen.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**
  Loop version.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxNgbBernsen** (**HxTagList** &tags)
  Constructor.
- ~**HxNgbBernsen** ()
- **HxSizes size** ()
- void **init** (int, int, ArgType)
- void **next** (int x, int y, ArgType value)
- ResType **result** () const

Static Public Methods

- **HxString className** ()
  The name : "bernsen".

8.358.1 Detailed Description

template<class ArgType, class ResType> class HxNgbBernsen< ArgType, ResType >

Neighbourhood functor for bernsen thresholding.

8.358.2 Member Typedef Documentation

8.358.2.1 template<class ArgType, class ResType> typedef HxTagLoop HxNgbBernsen::IteratorCategory

Loop version.

8.358.2.2 template<class ArgType, class ResType> typedef HxTag1Phase HxNgbBernsen::PhaseCategory

1 phase.

8.358.3 Constructor & Destructor Documentation

8.358.3.1 template<class ArgType, class ResType> HxNgbBernsen< ArgType, ResType >::HxNgbBernsen (HxTagList & *tags*)

Constructor.

Taglist should contain: int ”conn”, int ”wshedval”

```
25     : _values(0)
26 {
27     _size       = HxGetTag(tags, "windowSz", 31);
28     _uniformTh  = HxGetTag(tags, "uniformTh", 15);
29     _uniformLow = HxGetTag(tags, "uniformLow", true);
30     _conn       = HxGetTag(tags, "conn", 8);
31     _pctIdx     = 0.5 * (_size * _size);
32
33     _values = new ArgType[_size * _size];
34
35 }
```

8.358.4 Member Function Documentation

8.358.4.1 template<class ArgType, class ResType> HxString HxNgbBernsen< ArgType, ResType >::className () [inline, static]

The name : ”bernsen”.

```
73 {
74     static HxString s("bernsen");
75     return s;
76 }
```

The documentation for this class was generated from the following files:

- **HxNgbBernsen.h**
- HxNgbBernsen.c

8.359 HxNgbDefuz Class Template Reference

Neighbourhood functor for Defuz.

`#include <HxNgbDefuz.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**

  Loop version.

- typedef **HxTag1Phase PhaseCategory**

  1 phase.

Public Methods

- **HxNgbDefuz** (**HxTagList** &tags)

  Constructor.

- ~**HxNgbDefuz** ()
- **HxSizes size** ()
- void **init** (int, int, ArgType)
- void **next** (int x, int y, ArgType value)
- ResType **result** () const

Static Public Methods

- **HxString className** ()

  The name : "Defuz".

8.359.1 Detailed Description

template<class ArgType, class ResType> class HxNgbDefuz< ArgType, ResType >

Neighbourhood functor for Defuz.

8.359.2 Member Typedef Documentation

8.359.2.1 template<class ArgType, class ResType> typedef HxTagLoop HxNgbDefuz::IteratorCategory

Loop version.

8.359.2.2 template<class ArgType, class ResType> typedef HxTag1Phase HxNgbDefuz::PhaseCategory

1 phase.

8.359.3 Constructor & Destructor Documentation

8.359.3.1 template<class ArgType, class ResType> HxNgbDefuz< ArgType, ResType >::HxNgbDefuz (HxTagList & *tags*)

Constructor.

Taglist should contain: int "windowSzX" int "windowSzY" int "thr"

```
113     : _values(0)
114 {
115
116     _sizeX      = HxGetTag(tags, "windowSzX", 5);
117     _sizeY      = HxGetTag(tags, "windowSzY", 5);
118     _uniformTh  = HxGetTag(tags, "thr", 0.5);
```

```
119     _pctIdx     = 0.5 * (_sizeX * _sizeY);
120
121     _values = new ArgType[_sizeX * _sizeY];
122
123 }
```

8.359.4 Member Function Documentation

8.359.4.1 template<class ArgType, class ResType> HxString HxNgbDefuz< ArgType, ResType >::className () [inline, static]

The name : ”Defuz”.

```
72 {
73     static HxString s("defuz");
74     return s;
75 }
```

The documentation for this class was generated from the following file:

- **HxNgbDefuz.h**

8.360 HxNgbHilditch Class Template Reference

Neighbourhood functor for Hilditch.

`#include <HxNgbHilditch.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**
  Loop version.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxNgbHilditch** (**HxTagList** &tags)
  Constructor.
- ∼**HxNgbHilditch** ()
- **HxSizes size** ()
- void **init** (int, int, ArgType)
- void **next** (int x, int y, ArgType value)
- ResType **result** ()

Static Public Methods

- **HxString className** ()

  The name : "Hilditch".

8.360.1 Detailed Description

template<class ArgType, class ResType> class HxNgbHilditch< ArgType, ResType >

Neighbourhood functor for Hilditch.

8.360.2 Member Typedef Documentation

8.360.2.1 template<class ArgType, class ResType> typedef HxTagLoop HxNgbHilditch::IteratorCategory

Loop version.

8.360.2.2 template<class ArgType, class ResType> typedef HxTag1Phase HxNgbHilditch::PhaseCategory

1 phase.

8.360.3 Constructor & Destructor Documentation

8.360.3.1 template<class ArgType, class ResType> HxNgbHilditch< ArgType, ResType >::HxNgbHilditch (HxTagList & *tags*)

Constructor.

```
25     : _tags(tags)
26 {
27     _size       = 5;//3;
28     _pctIdx     = 12;//4; //0.5 * (_size * _size);
29 
30     int _idxKernel[9]={ 6, 7, 8,11,12,13,16,17,18};
31     int _idxAp1[9]  ={ 7, 8,13,18,17,16,11, 6, 7};
32     int _idxAp2[9]  ={ 2, 3, 8,13,12,11, 6, 1, 2};
33     int _idxAp4[9]  ={ 8, 9,14,19,18,17,12, 7, 8};
34 
35     for(int j=0;j<9;j++)
36     {
37         idxKernel[j]=_idxKernel[j];
38         idxAp1[j]   =_idxAp1[j];
39         idxAp2[j]   =_idxAp2[j];
40         idxAp4[j]   =_idxAp4[j];
41     }
42 
43     _changed = false;
44 
45     _values = new ArgType[_size * _size];
46 
47 }
```

8.360.4 Member Function Documentation

8.360.4.1 template<class ArgType, class ResType> HxString HxNgbHilditch< ArgType, ResType >::className () [inline, static]

The name : ”Hilditch”.

```
77 {
78     static HxString s("hilditch");
79     return s;
80 }
```

The documentation for this class was generated from the following files:

- **HxNgbHilditch.h**
- HxNgbHilditch.c

8.361 HxNgbIsMaxGradDir2d Class Template Reference

Non maximum suppression in the direction of the gradient.

`#include <HxNgbIsMaxGradDir2d.h>`

Public Types

- typedef **HxTagCnum IteratorCategory**

  Coordinate enumerated version.

- typedef **HxTag1Phase PhaseCategory**

  1 phase.

- typedef **HxCnum CnumType**

  Coordinate enumerator type.

Public Methods

- **HxNgbIsMaxGradDir2d** (**HxTagList** &tags)

  Constructor.

- ∼**HxNgbIsMaxGradDir2d** ()

  Destructor.

- **HxSizes size** ()

  Size of the neighbourhood.

- **CnumType begin** ()

  The first of the coordinates.

- **CnumType end** ()
  The last of the coordinates.
- void **init** (int x, int y, const ArithT &value)
  Initialization.
- void **next** (int x, int y, const ArithT &value)
  Processing one pixel.
- ResultT **result** () const
  Produce the result value.

Static Public Methods

- **HxString className** ()
  The name : "isMaxGradDir".

8.361.1 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.

8.361.2 Member Typedef Documentation

8.361.2.1 template<class ResultT, class ArithT> typedef HxTagCnum HxNgbIsMaxGradDir2d::IteratorCategory

Coordinate enumerated version.

8.361.2.2 template<class ResultT, class ArithT> typedef HxTag1Phase HxNgbIsMaxGradDir2d::PhaseCategory

1 phase.

8.361.2.3 template<class ResultT, class ArithT> typedef HxCnum HxNgbIsMaxGradDir2d::CnumType

Coordinate enumerator type.

8.361.3 Constructor & Destructor Documentation

8.361.3.1 template<class ResultT, class ArithT> HxNgbIsMaxGradDir2d< ResultT, ArithT >::HxNgbIsMaxGradDir2d (HxTagList & *tags*)

Constructor.

```
28 {
29     _coords = &_coordPairs[0];
30     _level = HxScalarDouble(HxGetTag<HxValue>(tags, "level", HxValue(3.0))).x();
31     if (_level < 0)
32         _level = 0;
33     _level *= _level;
34 }
```

8.361.3.2 template<class ResultT, class ArithT> HxNgbIsMaxGradDir2d< ResultT, ArithT >::~HxNgbIsMaxGradDir2d ()

Destructor.

```
38 {
39 }
```

8.361.4 Member Function Documentation

8.361.4.1 template<class ResultT, class ArithT> HxSizes HxNgbIsMaxGradDir2d< ResultT, ArithT >::size () [inline]

Size of the neighbourhood.

```
137 {
138     return HxSizes(3, 3, 1);
139 }
```

8.361.4.2 template<class ResultT, class ArithT> HxNgbIsMaxGradDir2d< ResultT, ArithT >::CnumType HxNgbIsMaxGradDir2d< ResultT, ArithT >::begin () [inline]

The first of the coordinates.

```
75 {
76     return CnumType(&_coords[0]);
77 }
```

8.361.4.3 template<class ResultT, class ArithT> HxNgbIsMaxGradDir2d< ResultT, ArithT >::CnumType HxNgbIsMaxGradDir2d< ResultT, ArithT >::end () [inline]

The last of the coordinates.

```
82 {
83     return CnumType(&_coords[_endIdx]);
84 }
```

8.361.4.4 template<class ResultT, class ArithT> void HxNgbIsMaxGradDir2d< ResultT, ArithT >::init (int *x*, int *y*, const ArithT & *v*) [inline]

Initialization.

```
89 {
90     static const double tan22_5 = 0.41421356;
91     static const double tan67_5 = 2.41421356;
92
93     double x(v.x()), y(v.y());
94     _resultMag = x*x+y*y;
95
96     if (_resultMag < _level)
97     {
98         _endIdx = 0;
99         return;
100     }
101
102     if (x == 0)
103     {
104         _endIdx = 2;
105         _coords = &_coordPairs[4];
106         return;
107     }
108
109     double rc = y/x;
110     double frc = fabs(rc);
111     int i = (frc < tan22_5) ? 0 : ( (frc < tan67_5) ? 1 : 2 );
112     if (rc < 0)
113         i += 3;
114     _endIdx = 2;
115     _coords = &_coordPairs[i<<1];
116 }
```

8.361.4.5 template<class ResultT, class ArithT> void HxNgbIsMaxGradDir2d< ResultT, ArithT >::next (int *x*, int *y*, const ArithT & *v*) [inline]

Processing one pixel.

```
129 {
130     if ((v.x()*v.x() + v.y()*v.y()) > _resultMag)
131         _resultMag = 0;
132 }
```

8.361.4.6 template<class ResultT, class ArithT> ResultT HxNgbIsMaxGradDir2d< ResultT, ArithT >::result () const [inline]

Produce the result value.

```
144 {
145     return _resultMag >= _level ? 1 : 0;
146 }
```

8.361.4.7 template<class ResultT, class ArithT> HxString HxNgbIsMaxGradDir2d< ResultT, ArithT >::className () [inline, static]

The name : "isMaxGradDir".

```
121 {
122     static HxString s("isMaxGradDir");
123     return s;
124 }
```

The documentation for this class was generated from the following files:

- **HxNgbIsMaxGradDir2d.h**
- HxNgbIsMaxGradDir2d.c

8.362 HxNgbKuwahara Class Template Reference

Neighbourhood functor for Kuwahara thresholding.

`#include <HxNgbKuwahara.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**

  Loop version.

- typedef **HxTag1Phase PhaseCategory**

  1 phase.

Public Methods

- **HxNgbKuwahara** (**HxTagList** &tags)

  Constructor.

- ~**HxNgbKuwahara** ()
- **HxSizes size** ()
- void **init** (int, int, ArgType)
- void **next** (int x, int y, ArgType value)
- ResType **result** () const

Static Public Methods

- **HxString className** ()

  The name : "Kuwahara".

8.362.1 Detailed Description

template<class ArgType, class ResType> class HxNgbKuwahara< ArgType, ResType >

Neighbourhood functor for Kuwahara thresholding.

8.362.2 Member Typedef Documentation

8.362.2.1 template<class ArgType, class ResType> typedef HxTagLoop HxNgbKuwahara::IteratorCategory

Loop version.

8.362.2.2 template<class ArgType, class ResType> typedef HxTag1Phase HxNgbKuwahara::PhaseCategory

1 phase.

8.362.3 Constructor & Destructor Documentation

8.362.3.1 template<class ArgType, class ResType> HxNgbKuwahara< ArgType, ResType >::HxNgbKuwahara (HxTagList & *tags*)

Constructor.

Taglist should contain: int ”windowW”, int ”windowH”

```
25     : _values(0)
26 {
27     _windowW    = HxGetTag(tags, "windowW", 3);
28     _windowH    = HxGetTag(tags, "windowH", 3);
29
30     _pctIdx     = 0.5 * (_windowW*_windowH);
31
32     _values = new ArgType[_windowW*_windowH];
33
34 }
```

8.362.4 Member Function Documentation

8.362.4.1 template<class ArgType, class ResType> HxString HxNgbKuwahara< ArgType, ResType >::className () [inline, static]

The name : ”Kuwahara”.

```
71 {
72     static HxString s("kuwahara");
73     return s;
74 }
```

The documentation for this class was generated from the following files:

- **HxNgbKuwahara.h**
- HxNgbKuwahara.c

8.363 HxNgbLocalMode Class Template Reference

Neighbourhood functor for local mode filter.

`#include <HxNgbLocalMode.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**
  Loop version.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxNgbLocalMode** (**HxTagList** &tags)
  Constructor.
- **HxSizes size** ()
  Size of the neighbourhood.
- void **init** (int, int, const SrcT &, const SrcT &v2)
  Initialization.
- void **check** (int i, int j)
- void **next** (int x, int y, const SrcT &v1, const SrcT &)
  Processing one pixel.
- DstT **result** () const
  Produce the result value.

Static Public Methods

- **HxString className** ()
  The name : "localMode".

8.363.1 Detailed Description

template<class DstT, class SrcT> class HxNgbLocalMode< DstT, SrcT >

Neighbourhood functor for local mode filter.

8.363.2 Member Typedef Documentation

8.363.2.1 template<class DstT, class SrcT> typedef HxTagLoop HxNgbLocalMode::IteratorCategory

Loop version.

8.363.2.2 template<class DstT, class SrcT> typedef HxTag1Phase HxNgbLocalMode::PhaseCategory

1 phase.

8.363.3 Constructor & Destructor Documentation

8.363.3.1 template<class DstT, class SrcT> HxNgbLocalMode< DstT, SrcT >::HxNgbLocalMode (HxTagList & *tags*) [inline]

Constructor.

```
39                              {
40                                  _size = HxGetTag(tags, "ngbSize", HxSizes(11, 11, 1));
41                                  if (_size.x()<1&&_size.x()%2!=0&&_size.y()<1&&_size.y()%2!=0&&_size.z()!=1)
42                                      throw HxString("Error: HxNgbLocalMode kernel must be 2D odd sized");
43                                  hsx = (_size.x()+0)/2; // e.g. s=11, half=5, xin=[0..11> xwork=[-5..+5]
44                                  hsy = (_size.y()+0)/2;
45                                  _tags=tags;
46                                  double sx = HxGetTag(tags, "sigmax", 2.0);
47                                  double sy = HxGetTag(tags, "sigmay", 2.0);
48                                  double sval = HxGetTag(tags, "sigmaval", 10.0);
49                                  _sigmax = -0.5/sx/sx;
50                                  _sigmay = -0.5/sy/sy;
51                                  _sigmaval = -0.5/sval/sval;
52                              }
```

8.363.4 Member Function Documentation

8.363.4.1 template<class DstT, class SrcT> HxSizes HxNgbLocalMode< DstT, SrcT >::size () [inline]

Size of the neighbourhood.

```
56                              {
57                                  return _size;
58                              }
```

8.363.4.2 template<class DstT, class SrcT> void HxNgbLocalMode< DstT, SrcT >::init (int, int, const SrcT &, const SrcT & *v2*) [inline]

Initialization.

```
62                              {
63                                  _v2=v2;
```

```
64                                  teller = HxScalarDouble(0.0);
65                                  noemer=0.0;
66                              }
```

8.363.4.3 template<class DstT, class SrcT> void HxNgbLocalMode< DstT, SrcT >::next (int *x*, int *y*, const SrcT & *v1*, const SrcT &) [inline]

Processing one pixel.

```
76                              {
77                                  int i=x-hsx;
78                                  int j=y-hsy;
79 #ifdef _DEBUG
80                                  check(i,j);
81 #endif
82                                  double po=0;
83                                  po += i*i*_sigmax;
84                                  po += j*j*_sigmay;
85                                  po += norm2sqr(v1-_v2)*_sigmaval;
86                                  double w=exp(po);
87                                  teller += v1*DstT(HxScalarDouble(w));
88                                  noemer += w;
89                              }
```

8.363.4.4 template<class DstT, class SrcT> DstT HxNgbLocalMode< DstT, SrcT >::result () const [inline]

Produce the result value.

```
93                              {
94                                  return teller/DstT(HxScalarDouble(noemer));
95                              }
```

8.363.4.5 template<class DstT, class SrcT> HxString HxNgbLocalMode< DstT, SrcT >::className () [inline, static]

The name : ”localMode”.

```
99                              {
100                                  return HxString("localMode");
101                             }
```

The documentation for this class was generated from the following file:

- **HxNgbLocalMode.h**

8.364 HxNgbLocalModeInst Class Template Reference

Instantiator for neighbourhood operation with local mode.

Public Attributes

- **HxImgFtorNgb2dExtra**< InOutT, InOutT, InOutT, **HxNgbLocalMode**< typename InOutT::ArithType, typename InOutT::ArithType > > **f**

  Instantiate image functor.

8.364.1 Detailed Description

template<**class InOutT**> **class HxNgbLocalModeInst**< **InOutT** >

Instantiator for neighbourhood operation with local mode.

8.364.2 Member Data Documentation

8.364.2.1 template<class InOutT> HxImgFtorNgb2dExtra< InOutT, InOutT, InOutT, HxNgbLocalMode<typename InOutT::ArithType, typename InOutT::ArithType> > HxNgbLocalModeInst::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbLocalModeInst.c

8.365 HxNgbLWshed2d Class Template Reference

Neighbourhood functor for watershed segmentation this look in a vecinity and decides if there is a border The smallest neighbour (if not wshed) is replaced by wshed value.

`#include <HxNgbLWshed2d.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**

  Loop version.

- typedef **HxTag1Phase PhaseCategory**

  1 phase.

Public Methods

- **HxNgbLWshed2d** (**HxTagList** &tags)

  Constructor.

- ~**HxNgbLWshed2d** ()

  Destructor.

- **HxSizes size** ()
  Size of the neighbourhood.
- void **init** (int, int, ArgType)
  Initialization.
- void **next** (int x, int y, ArgType value)
  Processing one pixel.
- ResType **result** () const
  Produce the result value.

Static Public Methods

- **HxString className** ()
  The name : "lwshed".

8.365.1 Detailed Description

template<class ArgType, class ResType> class HxNgbLWshed2d< ArgType, ResType >

Neighbourhood functor for watershed segmentation this look in a vecinity and decides if there is a border The smallest neighbour (if not wshed) is replaced by wshed value.

8.365.2 Member Typedef Documentation

8.365.2.1 template<class ArgType, class ResType> typedef HxTagLoop HxNgbLWshed2d::IteratorCategory

Loop version.

8.365.2.2 template<class ArgType, class ResType> typedef HxTag1Phase HxNgbLWshed2d::PhaseCategory

1 phase.

8.365.3 Constructor & Destructor Documentation

8.365.3.1 template<class ArgType, class ResType> HxNgbLWshed2d< ArgType, ResType >::HxNgbLWshed2d (HxTagList & *tags*)

Constructor.

Taglist should contain: int "conn", int "wshedval"

```
25     : _values(0)
26 {
27     _size       = HxGetTag(tags, "size", 3);
28     _wshedval   = HxGetTag(tags, "wshedval", -3);
29     _conn       = HxGetTag(tags, "conn", 8);
30     _pctIdx     = 4;
31
32     _values = new ArgType[_size * _size];
33
34 }
```

8.365.3.2 template<class ArgType, class ResType> HxNgbLWshed2d< ArgType, ResType >::~HxNgbLWshed2d ()

Destructor.

```
38 {
39     if (_values)
40         delete [] _values;
41 }
```

8.365.4 Member Function Documentation

8.365.4.1 template<class ArgType, class ResType> HxSizes HxNgbLWshed2d< ArgType, ResType >::size () [inline]

Size of the neighbourhood.

```
105 {
106 //    return HxSizes(_size, _size, 1);
107     return HxSizes(3, 3, 1);
108 }
```

8.365.4.2 template<class ArgType, class ResType> void HxNgbLWshed2d< ArgType, ResType >::init (int, int, ArgType) [inline]

Initialization.

```
91 {
92     _i = 0;
93 }
```

8.365.4.3 template<class ArgType, class ResType> void HxNgbLWshed2d< ArgType, ResType >::next (int *x*, int *y*, ArgType *value*) [inline]

Processing one pixel.

```
98 {
99     _values[_i++] = value;
100 }
```

8.365.4.4 template<class ArgType, class ResType> ResType HxNgbLWshed2d< ArgType, ResType >::result () const [inline]

Produce the result value.

```
113 {
114     ArgType minval=_values[_pctIdx];
115     if(_conn==4)
116     {
117         if( _values[1] < minval && _values[1]>_wshedval)
118             minval = _values[1];
119         if( _values[3] < minval && _values[3]>_wshedval)
120             minval = _values[3];
121         if( _values[5] < minval && _values[5]>_wshedval)
122             minval = _values[5];
123         if( _values[7] < minval && _values[7]>_wshedval)
124             minval = _values[7];
125 
126         if(minval<_values[_pctIdx])
127             return _wshedval;
128         else
129             return _values[_pctIdx];
130     }
131     if(_conn==8)
132     {
133         if( _values[0] < minval && _values[0]>_wshedval)
134             minval = _values[0];
135         if( _values[1] < minval && _values[1]>_wshedval)
136             minval = _values[1];
137         if( _values[2] < minval && _values[2]>_wshedval)
138             minval = _values[2];
139         if( _values[3] < minval && _values[3]>_wshedval)
140             minval = _values[3];
141         if( _values[5] < minval && _values[5]>_wshedval)
142             minval = _values[5];
143         if( _values[6] < minval && _values[6]>_wshedval)
144             minval = _values[6];
145         if( _values[7] < minval && _values[7]>_wshedval)
146             minval = _values[7];
147         if( _values[8] < minval && _values[8]>_wshedval)
148             minval = _values[8];
149 
150         if(minval>_wshedval && minval<_values[_pctIdx])
151             return _wshedval;
152         else
153             return _values[_pctIdx];
154     }
155 
156     return _values[_pctIdx];
157 }
```

8.365.4.5 template<class ArgType, class ResType> HxString HxNgbLWshed2d< ArgType, ResType >::className () [inline, static]

The name : ”lwshed”.

```
83 {
84     static HxString s("lwshed");
85     return s;
86 }
```

The documentation for this class was generated from the following files:

- **HxNgbLWshed2d.h**
- HxNgbLWshed2d.c

8.366 HxNgbNonMaxSuppression2d Class Template Reference

Neighbourhood functor for non maximum suppression in the gradient direction.

`#include <HxNgbNonMaxSuppression2d.h>`

Public Types

- typedef **HxTagCnum IteratorCategory**
  Coordinate enumerated version.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.
- typedef **HxCnum CnumType**
  Coordinate enumerator type.

Public Methods

- **HxNgbNonMaxSuppression2d** (**HxTagList** &tags)
  Constructor.
- ~**HxNgbNonMaxSuppression2d** ()
  Destructor.
- **HxSizes size** ()
  Size of the neighbourhood.
- **CnumType begin** ()
  The first of the coordinates.
- **CnumType** & **end** ()
  The last of the coordinates.
- void **init** (int x, int y, const ArithT &value)
  Initialization.
- void **next** (int x, int y, const ArithT &value)
  Processing one pixel.
- const ArithT & **result** () const
  Produce the result value.

Static Public Methods

- **HxString className** ()
  The name : "nonMaxSuppression".

8.366.1 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.

8.366.2 Member Typedef Documentation

8.366.2.1 template<class ArithT> typedef HxTagCnum HxNgbNonMaxSuppression2d::IteratorCategory

Coordinate enumerated version.

8.366.2.2 template<class ArithT> typedef HxTag1Phase HxNgbNonMaxSuppression2d::PhaseCategory

1 phase.

8.366.2.3 template<class ArithT> typedef HxCnum HxNgbNonMaxSuppression2d::CnumType

Coordinate enumerator type.

8.366.3 Constructor & Destructor Documentation

8.366.3.1 template<class ArithT> HxNgbNonMaxSuppression2d< ArithT >::HxNgbNonMaxSuppression2d (HxTagList & *tags*)

Constructor.

```
28     : _nullPix(HxScalarInt(0))
29 {
30     _coords = &_coordPairs[0];
31 }
```

8.366.3.2 template<class ArithT> HxNgbNonMaxSuppression2d< ArithT >::~HxNgbNonMaxSuppression2d ()

Destructor.

```
35 {
36 }
```

8.366.4 Member Function Documentation

8.366.4.1 template<class ArithT> HxSizes HxNgbNonMaxSuppression2d< ArithT >::size () [inline]

Size of the neighbourhood.

```
137 {
138     return HxSizes(3, 3, 1);
139 }
```

8.366.4.2 template<class ArithT> TYPENAME HxNgbNonMaxSuppression2d< ArithT >::CnumType HxNgbNonMaxSuppression2d< ArithT >::begin () [inline]

The first of the coordinates.

```
79 {
80     return CnumType(&_coords[0]);
81 }
```

8.366.4.3 template<class ArithT> TYPENAME HxNgbNonMaxSuppression2d< ArithT >::CnumType & HxNgbNonMaxSuppression2d< ArithT >::end () [inline]

The last of the coordinates.

```
86 {
87     return _end;
88 }
```

8.366.4.4 template<class ArithT> void HxNgbNonMaxSuppression2d< ArithT >::init (int *x*, int *y*, const ArithT & *v*) [inline]

Initialization.

```
95 {
96     HxBreakPoint();
97     static const double tan22_5 = 0.41421356;
98     static const double tan67_5 = 2.41421356;
99
100     double x(v.x()), y(v.y());
101     _resultPix = v;
102     _resultMag = x*x+y*y;
103     _suppressed = false;
104
105     if (x != 0) {
106         double rc = y/x;
107         double frc = fabs(rc);
108         int i = (frc < tan22_5) ? 0 : ( (frc < tan67_5) ? 1 : 2 );
109         if (rc < 0)
110             i += 3;
111         _coords = &_coordPairs[i<<1];
112     } else {
113         _coords = &_coordPairs[4];
114     }
115     _end = &_coords[2];
116 }
```

8.366.4.5 template<class ArithT> void HxNgbNonMaxSuppression2d< ArithT >::next (int *x*, int *y*, const ArithT & *v*) [inline]

Processing one pixel.

```
129 {
130     if ((v.x()*v.x() + v.y()*v.y()) > _resultMag)
131         _suppressed = true;
132 }
```

8.366.4.6 template<class ArithT> const ArithT & HxNgbNonMaxSuppression2d< ArithT >::result () const [inline]

Produce the result value.

```
144 {
145     return _suppressed ? _nullPix : _resultPix;
146 }
```

8.366.4.7 template<class ArithT> HxString HxNgbNonMaxSuppression2d< ArithT >::className () [inline, static]

The name : ”nonMaxSuppression”.

```
121 {
122     static HxString s("nonMaxSuppression");
123     return s;
124 }
```

The documentation for this class was generated from the following files:

- **HxNgbNonMaxSuppression2d.h**
- HxNgbNonMaxSuppression2d.c

8.367 HxNgbOpticalFlowInst Class Template Reference

Instantiator for neighbourhood operation with TalkNgbExtra2P1Cnum.

Public Attributes

- **HxImgFtorNgb2dExtra2**< ResSigT, ImgSigT, ImgSigT, ImgSigT, HxNgbOpticalFlow< typename ResSigT::ArithTypeDouble, typename ImgSigT::ArithType > > **f**

  Instantiate image functor.

8.367.1 Detailed Description

template<class ResSigT, class ImgSigT> class HxNgbOpticalFlowInst< ResSigT, ImgSigT >

Instantiator for neighbourhood operation with TalkNgbExtra2P1Cnum.

8.367.2 Member Data Documentation

8.367.2.1 template<class ResSigT, class ImgSigT> HxImgFtorNgb2dExtra2< ResSigT, ImgSigT, ImgSigT, ImgSigT, HxNgbOpticalFlow<typename ResSigT::ArithTypeDouble, typename ImgSigT::ArithType> > HxNgbOpticalFlowInst::f

Instantiate image functor.

The documentation for this class was generated from the following file:

- HxNgbOpticalFlowInst.c

8.368 HxNgbPercentile2d Class Template Reference

Neighbourhood functor for percentile filter.

`#include <HxNgbPercentile2d.h>`

Public Types

- typedef **HxTagLoop IteratorCategory**
  Loop version.
- typedef **HxTag1Phase PhaseCategory**
  1 phase.

Public Methods

- **HxNgbPercentile2d** (**HxTagList** &tags)
  Constructor.
- ~**HxNgbPercentile2d** ()
  Destructor.
- **HxSizes size** ()
  Size of the neighbourhood.
- void **init** (int, int, ArithT)
  Initialization.
- void **next** (int x, int y, ArithT value)
  Processing one pixel.
- ArithT **result** () const
  Produce the result value.

Static Public Methods

- **HxString className** ()

  The name : "percentile".

8.368.1 Detailed Description

template<class ArithT> class HxNgbPercentile2d< ArithT >

Neighbourhood functor for percentile filter.

8.368.2 Member Typedef Documentation

8.368.2.1 template<class ArithT> typedef HxTagLoop HxNgbPercentile2d::IteratorCategory

Loop version.

8.368.2.2 template<class ArithT> typedef HxTag1Phase HxNgbPercentile2d::PhaseCategory

1 phase.

8.368.3 Constructor & Destructor Documentation

8.368.3.1 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 }
```

8.368.3.2 template<class ArithT> HxNgbPercentile2d< ArithT >::~HxNgbPercentile2d ()

Destructor.

```
32 {
33     if (_values)
34         delete [] _values;
35 }
```

8.368.4 Member Function Documentation

8.368.4.1 template<class ArithT> HxSizes HxNgbPercentile2d< ArithT >::size () [inline]

Size of the neighbourhood.

```
97 {
98     return HxSizes(_size, _size, 1);
99 }
```

8.368.4.2 template<class ArithT> void HxNgbPercentile2d< ArithT >::init (int, int, ArithT) [inline]

Initialization.

```
83 {
84     _i = 0;
85 }
```

8.368.4.3 template<class ArithT> void HxNgbPercentile2d< ArithT >::next (int *x*, int *y*, ArithT *value*) [inline]

Processing one pixel.

```
90 {
91     _values[_i++] = value;
92 }
```

8.368.4.4 template<class ArithT> ArithT HxNgbPercentile2d< ArithT >::result () const [inline]

Produce the result value.

```
104 {
105     std::sort(&_values[0], &_values[_size*_size]);
106     return _values[_pctIdx];
107 }
```

8.368.4.5 template<class ArithT> HxString HxNgbPercentile2d< ArithT >::className () [inline, static]

The name : ”percentile”.

```
75 {
76     static HxString s("percentile");
77     return s;
78 }
```

The documentation for this class was generated from the following files:

- **HxNgbPercentile2d.h**
- HxNgbPercentile2d.c

8.369 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).

8.369.1 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.

8.369.2 Constructor & Destructor Documentation

8.369.2.1 HxNJet::HxNJet ()

Constructor.

```
15                : _pointee(0)
16 {
17 }
```

8.369.2.2 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 }
```

8.369.2.3 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 }
```

8.369.2.4 HxNJet::HxNJet (const HxNJet & *rhs*)

Copy constructor.

```
35                                  : _order(rhs._order), _scale(rhs._scale),
36                                      _pointee(rhs.pointee())
37 {
38 }
```

8.369.2.5 HxNJet::~HxNJet () [virtual]

Destructor.

```
41 {
42 }
```

8.369.3 Member Function Documentation

8.369.3.1 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 }
```

8.369.3.2 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 }
```

8.369.3.3 int HxNJet::ident () const

The identity of the NJet.

```
63 {
64     return pointee() ? pointee()->ident() : 0;
65 }
```

8.369.3.4 int HxNJet::order () const

The order N.

```
69 {
70     return _order;
71 }
```

8.369.3.5 double HxNJet::scale () const

The scale.

```
75 {
76     return _scale;
77 }
```

8.369.3.6 int HxNJet::nrComponents () const

The number of components.

```
81 {
82     return pointee() ? pointee()->nrComponents() : 0;
83 }
```

8.369.3.7 int HxNJet::isColor () const

Indicator whether its a color or grey value NJet.

```
87 {
88     return pointee() ? pointee()->isColor() : 0;
89 }
```

8.369.3.8 HxImageRep HxNJet::xy (int *x*, int *y*) const

Get the specified 2D component.

```
114 {
115     return getLidx(ord2idx(x, y));
116 }
```

8.369.3.9 HxImageRep HxNJet::xyz (int *x*, int *y*, int *z*) const

Get the specified 3D component.

```
120 {
121     return getLidx(ord2idx(x, y, z));
122 }
```

8.369.3.10 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 }
```

8.369.3.11 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 }
```

8.369.3.12 HxImageRep HxNJet::getLidx (int *i*) const

Get the specified L component.

```
93 {
94     HxImageRep im;
95     return pointee() ? pointee()->getL(i) : im;
96 }
```

8.369.3.13 HxImageRep HxNJet::getJidx (int *i*) const

Get the specified J component.

```
100 {
101     HxImageRep im;
102     return pointee() ? pointee()->getJ(i) : im;
103 }
```

8.369.3.14 HxImageRep HxNJet::getMidx (int *i*) const

Get the specified M component.

```
107 {
108     HxImageRep im;
109     return pointee() ? pointee()->getM(i) : im;
110 }
```

8.369.3.15 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 }
```

8.369.3.16 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 }
```

8.369.3.17 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 }
```

8.369.3.18 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 }
```

8.369.3.19 HxImageRep HxNJet::getLw () const

Get the L gradient magnitude.

```
206 {
207     HxImageRep im;
208     return pointee() ? pointee()->getLw() : im;
209 }
```

8.369.3.20 HxImageRep HxNJet::getJw () const

Get the J gradient magnitude.

```
213 {
214     HxImageRep im;
215     return pointee() ? pointee()->getJw() : im;
216 }
```

8.369.3.21 HxImageRep HxNJet::getMw () const

Get the M gradient magnitude.

```
220 {
221     HxImageRep im;
222     return pointee() ? pointee()->getMw() : im;
223 }
```

8.369.3.22 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 }
```

8.369.3.23 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;}
```

8.369.3.24 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

8.370 HxPixelAllocator Class Template Reference

Class for (logged) allocation and deallocation of pixel data.

`#include <HxPixelAllocator.h>`

Public Types

- typedef PixelT * **pointer**
- typedef const PixelT * **const_pointer**
- typedef PixelT & **reference**
- typedef const PixelT & **const_reference**
- typedef PixelT **value_type**
- typedef size_t **size_type**

Public Methods

- **HxPixelAllocator** ()
- ~**HxPixelAllocator** ()
- pointer **address** (reference x)
- const_pointer **const_address** (const_reference x)
- pointer **allocate** (size_type n)
- void **deallocate** (pointer p, size_type n=0)

8.370.1 Detailed Description

template<class PixelT> class HxPixelAllocator< PixelT >

Class for (logged) allocation and deallocation of pixel data.

The documentation for this class was generated from the following files:

- **HxPixelAllocator.h**
- HxPixelAllocator.c

8.371 HxPointAndValue Class Reference

this is used to make the image as alist that can be sorted by pixel values.

Public Methods

- **HxPointAndValue** ()
- **HxPointAndValue** (HxPoint2 p1, **HxScalarInt** v1)
- bool **operator**< (const HxPointAndValue &pv) const

Public Attributes

- HxPoint2 **p**
- **HxScalarInt v**

8.371.1 Detailed Description

this is used to make the image as alist that can be sorted by pixel values.

The documentation for this class was generated from the following file:

- HxWatershedSlow.c

8.372 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.

8.372.1 Detailed Description

Class definition for list of HxPoint's.

8.372.2 Member Typedef Documentation

8.372.2.1 typedef std::back_insert_iterator<HxPointList> HxPointList::back_insert_iterator

back inserter.

8.372.3 Member Function Documentation

8.372.3.1 HxPointList & HxPointList::operator<< (const HxPoint & *s*) [inline]

Add point to the list.

```
47 {
48     push_back(s);
49     return *this;
50 }
```

8.372.3.2 void HxPointList::eraseAll () [inline]

Remove all points from the list.

```
54 {
55     erase(begin(), end());
56 }
```

The documentation for this class was generated from the following file:

- **HxPointList.h**

8.373 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**

8.373.1 Detailed Description

Class definition for points in R2 (real-value coordinates).

8.373.2 Constructor & Destructor Documentation

8.373.2.1 HxPointR2::HxPointR2 () `[inline]`

Constructor.

```
68                      : _data(0,0)
69 {
70 }
```

8.373.2.2 HxPointR2::HxPointR2 (double *d1*, double *d2*) [inline]

Constructor.

```
73                                          : _data(d1, d2)
74 {
75 }
```

8.373.2.3 HxPointR2::HxPointR2 (const HxVec2Double & *v*) [inline]

Copy constructor.

```
78                                           : _data(v)
79 {
80 }
```

8.373.3 Member Function Documentation

8.373.3.1 double HxPointR2::x () const [inline]

Get the x coordinate of the point.

```
84 {
85     return _data.x();
86 }
```

8.373.3.2 double HxPointR2::y () const [inline]

Get the y coordinate of the point.

```
90 {
91     return _data.y();
92 }
```

8.373.3.3 HxPointR2 HxPointR2::add (const HxVectorR2 & *arg*) const

Add the given vector to this point.

```
16 {
17     return HxPointR2(_data + arg._data );
18
19 }
```

8.373.3.4 HxPointR2 HxPointR2::sub (const HxVectorR2 & *arg*) const

Subtract the given vector from this point.

```
23 {
24     return HxPointR2(_data - arg._data);
25 }
```

8.373.3.5 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:

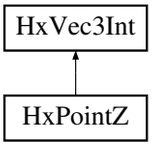
- **HxPointR2.h**
- HxPointR2.c

8.374 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.

8.374.1 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.

8.374.2 Constructor & Destructor Documentation

8.374.2.1 HxPointZ::HxPointZ (int *x* = 0, int *y* = 0, int *z* = 0) [inline]

Constructor.

```
34                                          : HxVec3Int(x, y, z)
35 {
36 }
```

8.374.2.2 HxPointZ::HxPointZ (const HxVec3Int & *rhs*) [inline]

Constructor.

```
40     : HxVec3Int(rhs.x(), rhs.y(), rhs.z())
41 {
42 }
```

8.374.2.3 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**

8.375 HxPointZList Class Reference

Class definition for list of **HxPointZ** (p. 1077)'s.

`#include <HxPointZList.h>`

Public Methods

- HxPointZList & **operator**<< (const **HxPointZ** &)
  Add point to the list.
- void **eraseAll** ()
  Remove all points from the list.

8.375.1 Detailed Description

Class definition for list of **HxPointZ** (p. 1077)'s.

Specialization of list from STL.

8.375.2 Member Function Documentation

8.375.2.1 HxPointZList & HxPointZList::operator<< (const HxPointZ & *s*) [inline]

Add point to the list.

```
97 {
98     push_back(s);
99     return *this;
100 }
```

8.375.2.2 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**

8.376 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.

8.376.1 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.

8.376.2 Constructor & Destructor Documentation

8.376.2.1 HxPolyline2d::HxPolyline2d () [inline]

Construct an empty polyline.

```
80 {
81     _ident = _nr++;
82 }
```

8.376.2.2 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 }
```

8.376.2.3 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 }
```

8.376.2.4 HxPolyline2d::~HxPolyline2d () [inline]

Destructor.

```
102 {
103 }
```

8.376.3 Member Function Documentation

8.376.3.1 int HxPolyline2d::ident () const [inline]

Get the identifier of this object.

```
107 {
108     return _ident;
109 }
```

8.376.3.2 HxPointSetR2 HxPolyline2d::getPoints () const [inline]

Get the points of the object.

```
113 {
114     return _points;
115 }
```

8.376.3.3 int HxPolyline2d::getClosed () const [inline]

Indicate whether this is a polyline or polygon.

```
119 {
120     return _closed;
121 }
```

8.376.3.4 int HxPolyline2d::getNrPoints () const [inline]

Get the number of points.

```
125 {
126     return _points.size();
127 }
```

8.376.3.5 HxPointR2 HxPolyline2d::getPoint (int *index*) const [inline]

Get the points at the given index.

```
131 {
132     return _points[index];
133 }
```

8.376.3.6 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 }
```

8.376.3.7 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:

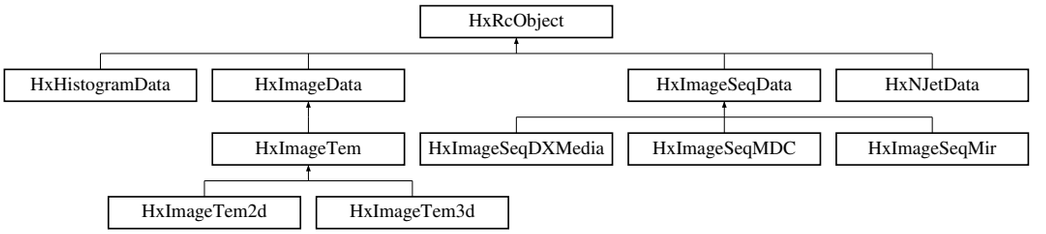
- **HxPolyline2d.h**
- HxPolyline2d.c

8.377 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** ()

8.377.1 Detailed Description

Base class for reference counted objects.

Use with **HxRcPtr** (p. 1083).

The documentation for this class was generated from the following files:

- **HxRcObject.h**
- HxRcObject.c

8.378 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

8.378.1 Detailed Description

template<class T> class HxRcPtr< T >

Template class for (smart) pointers to reference counted objects, i.e.

objects derived from **HxRcObject** (p. 1082).

The documentation for this class was generated from the following file:

- **HxRcPtr.h**

8.379 HxRegData Class Reference

A registry value.

`#include <HxRegData.h>`

Public Types

- enum **RvType** { **Int**, **String** }
  The type of the data.

Public Methods

- **HxRegData** ()
  Constructor.
- **HxRegData** (**HxString**)
  Constructor.
- **HxRegData** (int)
  Constructor.
- **HxRegData** (const HxRegData &)
  Constructor.
- ~**HxRegData** ()
  Desstructor.

- HxRegData & **operator=** (const HxRegData &)

  Assignment operator.

- **RvType type** () const

  Get type of data.

- void **setString** (**HxString**)

  Set string data.

- **HxString getString** () const

  Get string data.

- void **setInt** (int)

  Set int data.

- int **getInt** () const

  Get int data.

- **HxString toString** () const

  Output to string.

- STD_OSTREAM & **put** (STD_OSTREAM &, int cCode=0) const

  Put on stream.

8.379.1 Detailed Description

A registry value.

8.379.2 Member Enumeration Documentation

8.379.2.1 enum HxRegData::RvType

The type of the data.

```
41 { Int, String } RvType;
```

8.379.3 Constructor & Destructor Documentation

8.379.3.1 HxRegData::HxRegData () [inline]

Constructor.

```
84 {
85     _type = String;
86     new((void*)&(_val._str[0])) HxPnString(HxString());
87 }
```

8.379.3.2 HxRegData::HxRegData (HxString *s*) [inline]

Constructor.

```
91 {
92     _type = String;
93     new((void*)&(_val._str[0])) HxPnString(s);
94 }
```

8.379.3.3 HxRegData::HxRegData (int *i*) [inline]

Constructor.

```
98 {
99     _type = Int;
100     _val._int = i;
101 }
```

8.379.3.4 HxRegData::HxRegData (const HxRegData & *rhs*)

Constructor.

```
15 {
16     _type = rhs._type;
17     switch (_type) {
18     case Int    :
19         _val._int = rhs._val._int;
20         break;
21     case String :
22         new((void*)&(_val._str[0]))
23             HxPnString(((HxPnString*)&(rhs._val._str[0]))->toString());
24         break;
25     }
26 }
```

8.379.3.5 HxRegData::~HxRegData () [inline]

Desstructor.

```
105 {
106     if (_type == String)
107         ((HxPnString*)&(_val._str[0]))->~HxPnString();
108 }
```

8.379.4 Member Function Documentation

8.379.4.1 HxRegData & HxRegData::operator= (const HxRegData & *rhs*)

Assignment operator.

```
30 {
31     if (_type == String)
32         ((HxPnString*)&(_val._str[0]))->~HxPnString();
33     _type = rhs._type;
34     switch (_type) {
35     case Int    :
36         _val._int = rhs._val._int;
37         break;
38     case String :
39         new((void*)&(_val._str[0]))
40             HxPnString(((HxPnString*)&(rhs._val._str[0]))->toString());
41         break;
42     }
43     return *this;
44 }
```

8.379.4.2 HxRegData::RvType HxRegData::type () const [inline]

Get type of data.

```
112 {
113     return _type;
114 }
```

8.379.4.3 void HxRegData::setString (HxString *s*) [inline]

Set string data.

```
118 {
119     if (_type == String)
120         ((HxPnString*)&(_val._str[0]))->~HxPnString();
121     new((void*)&(_val._str[0])) HxPnString(s);
122     _type = String;
123 }
```

8.379.4.4 HxString HxRegData::getString () const [inline]

Get string data.

```
127 {
128     return (_type == String) ?
129                 ((HxPnString*)&(_val._str[0]))->toString() :
130                 HxString();
131 }
```

8.379.4.5 void HxRegData::setInt (int *i*) [inline]

Set int data.

```
135 {
136     if (_type == String)
137         ((HxPnString*)&(_val._str[0]))->~HxPnString();
138     _val._int = i;
139     _type = Int;
140 }
```

8.379.4.6 int HxRegData::getInt () const [inline]

Get int data.

```
144 {
145     return (_type == Int) ? _val._int : 0 ;
146 }
```

8.379.4.7 HxString HxRegData::toString () const

Output to string.

```
48 {
49     switch (_type) {
50     case Int    :
51         return makeString(_val._int);
52     case String :
53         return ((HxPnString*)&(_val._str[0]))->toString();
54     default     :
55         return HxString("\"") + __FILE__ + "\", line " + makeString(__LINE__)
56             + ": Implementation error";
57     }
58 }
```

8.379.4.8 STD_OSTREAM & HxRegData::put (STD_OSTREAM & *os*, int *cCode* = 0) const

Put on stream.

```
62 {
63     HxString quote = cCode ? "\\\"" : "\"";
64     switch (_type) {
65     case Int    :
66         os << _val._int;
67         break;
68     case String :
69         os << quote;
70 #ifdef HxRegData_Debug
71         (((HxPnString*)&(_val._str[0]))->toString()).dput(os);
72 #else
73         os << ((HxPnString*)&(_val._str[0]))->toString();
74 #endif
75         os << quote;
76         break;
77     }
78     return os;
79 }
```

The documentation for this class was generated from the following files:

- **HxRegData.h**
- HxRegData.c

8.380 HxRegistry Class Reference

The registry.

`#include <HxRegistry.h>`

Public Methods

- **HxRegistry** ()
  Constructor.
- ~**HxRegistry** ()
  Destructor.
- int **import** (**HxString** fileName)
  Import registry data from file.
- int **import** (const char *argv[ ])
  Import registry data from argument strings.
- int **exportText** (STD_OSTREAM &out) const
  Export registry data as text to stream.
- int **exportC** (STD_OSTREAM &out, **HxString** label) const
  Export registry data in format for use with **HxRegistryImporter** (p. 1095).
- **HxRegKey** * **insertKey** (**HxString** path)
  Insert key.
- void **eraseKey** (**HxString** path)
  Erase key.
- **HxRegKey** * **findKey** (**HxString** path) const
  Find key.
- void **insertValue** (**HxString** path, const **HxRegData** &)
  Insert value.
- void **insertValue** (**HxString** path, **HxString** data)
  Insert value.
- void **insertValue** (**HxString** path, int data)
  Insert value.
- void **eraseValue** (**HxString** path)
  Erase value.
- **HxString findValue** (**HxString** path) const
  Find value.
- int **valueExists** (**HxString** path) const
  Check whether value exists.
- **HxRegKey** * **setCursorKey** (**HxString** path)
  Set cursor key.

- **HxRegKey** * **setCursorUp** ()

  Move cursor key up.

- **HxRegKey** * **getCursorKey** () const

  Get cursor key.

- **HxString getCursorName** () const

  Get cursor name.

- **HxRegKey** * **setRootKey** ()

  Set root key.

- **HxRegKey** * **getRootKey** () const

  Get root key.

- STD_OSTREAM & **put** (STD_OSTREAM &) const

  Put registry on stream.

Static Public Methods

- HxRegistry & **instance** ()

  Access to the single instance of the registry.

8.380.1 Detailed Description

The registry.

8.380.2 Constructor & Destructor Documentation

8.380.2.1 HxRegistry::HxRegistry ()

Constructor.

```
28 {
29     _rootKey = HxRegKey::createRootKey();
30     _cursorKey = _rootKey;
31 }
```

8.380.2.2 HxRegistry::~HxRegistry ()

Destructor.

```
41 {
42     delete _rootKey;
43 }
```

8.380.3 Member Function Documentation

8.380.3.1 HxRegistry & HxRegistry::instance () [static]

Access to the single instance of the registry.

```
47 {
48     static HxRegistry theRegistry;
49     return theRegistry;
50 }
```

8.380.3.2 int HxRegistry::import (HxString *fileName*)

Import registry data from file.

```
88 {
89     STD_IFSTREAM inStream(fileName.c_str());
90
91     if (!inStream) {
92         HxEnvironment::instance()->errorStream()
93             << "Cannot open file " << fileName << STD_ENDL;
94         HxEnvironment::instance()->flush();
95         return 0;
96     }
97
98     HxStreamCharReader  reader(inStream);
99
100     HxRegParser::instance().setReader(reader);
101     HxRegParser::instance().setFileName(fileName);
102     HxRegParser::instance().setRegistry(*this);
103     HxRegParser::instance().parse();
104     return 1;
105 }
```

8.380.3.3 int HxRegistry::import (const char * *argv*[ ])

Import registry data from argument strings.

```
109 {
110     if (!*argv)
111         return 0;
112     HxString fileName(argv[0]);
113     fileName += "::";
114     fileName += argv[1];
115     argv += 2;
116     HxArgvCharReader reader(argv);
117
118     HxRegParser::instance().setReader(reader);
119     HxRegParser::instance().setFileName(fileName);
120     HxRegParser::instance().setRegistry(*this);
121     HxRegParser::instance().parse();
122     return 1;
123 }
```

8.380.3.4 int HxRegistry::exportText (STD_OSTREAM & *out*) const

Export registry data as text to stream.

```
127 {
128     _rootKey->put(out);
129     return 1;
130 }
```

8.380.3.5 int HxRegistry::exportC (STD_OSTREAM & *out*, HxString *label*) const

Export registry data in format for use with **HxRegistryImporter** (p. 1095).

```
134 {
135     out << "static const char* " << label << "[] = {" << STD_ENDL;
136     out << "__FILE__" << "," << STD_ENDL;
137     out << "\"" << label << "\"," << STD_ENDL;
138     _rootKey->put(out, "", hxTrue);
139     out << "(char*)0" << STD_ENDL << "};" << STD_ENDL;
140     return 1;
141 }
```

8.380.3.6 HxRegKey * HxRegistry::insertKey (HxString *name*)

Insert key.

```
145 {
146     HxStringList            nameList;
147     splitString(name, '/', std::back_inserter(nameList));
148     HxStringListConstIter   namePtr = nameList.begin();
149     HxRegKey*               key = _cursorKey;
150 
151     if ((namePtr != nameList.end()) && (*namePtr).empty()) {
152         key = _rootKey;
153         namePtr++;
154     }
155 
156     for (;namePtr != nameList.end(); namePtr++) {
157         if ((*namePtr).empty())
158             continue;
159         key = key->insertKey(*namePtr);
160     }
161     return key;
162 }
```

8.380.3.7 void HxRegistry::eraseKey (HxString *name*)

Erase key.

```
166 {
167     HxRegKey* key = findKey(name);
168     if (key && (key != _rootKey)) {
169         HxString baseName = key->getName();
170         key = key->getParent();
171         key->eraseKey(baseName);
172     }
173 }
```

8.380.3.8 HxRegKey * HxRegistry::findKey (HxString *name*) const

Find key.

```
177 {
178     HxRegKey*               key = _cursorKey;
179
180     if (!name.empty() && (name[0] == '/'))
181         key = _rootKey;
182
183     return key->findKey(name);
184 }
```

8.380.3.9 void HxRegistry::insertValue (HxString *path*, const HxRegData & *data*)

Insert value.

```
188 {
189     HxStringList nameList;
190     int n = splitString(path, '/', std::back_inserter(nameList));
191     if (n <= 0)
192         return;
193     HxString name = nameList.back();
194     if (name.empty())
195         return;
196     nameList.pop_back();
197     HxRegKey* key = _rootKey->findKey(nameList.begin(), nameList.end());
198     if (!key)
199         return;
200     key->insertValue(name, data);
201 }
```

8.380.3.10 void HxRegistry::insertValue (HxString *path*, HxString *data*) [inline]

Insert value.

```
109 {
110     insertValue(path, HxRegData(data));
111 }
```

8.380.3.11 void HxRegistry::insertValue (HxString *path*, int *data*) [inline]

Insert value.

```
115 {
116     insertValue(path, HxRegData(data));
117 }
```

8.380.3.12 void HxRegistry::eraseValue (HxString *path*)

Erase value.

```
237 {
238     HxStringList nameList;
239     int n = splitString(path, '/', std::back_inserter(nameList));
240     if (n <= 0)
241         return;
242     HxString name = nameList.back();
243     if (name.empty())
244         return;
245     nameList.pop_back();
246     HxRegKey* key = _rootKey->findKey(nameList.begin(), nameList.end());
247     if (key)
248         key->eraseValue(name);
249 }
```

8.380.3.13 HxString HxRegistry::findValue (HxString *path*) const

Find value.

```
220 {
221     const HxRegValue* val = doFindValue(path);
222     if (val)
223         return val->getData().toString();
224     else
225         return HxString("");
226 }
```

8.380.3.14 int HxRegistry::valueExists (HxString *path*) const

Check whether value exists.

```
230 {
231     const HxRegValue* val = doFindValue(path);
232     return val ? 1 : 0;
233 }
```

8.380.3.15 HxRegKey * HxRegistry::setCursorKey (HxString *name*)

Set cursor key.

```
253 {
254     HxRegKey* key = findKey(name);
255     if (key)
256         _cursorKey = key;
257     return key;
258 }
```

8.380.3.16 HxRegKey * HxRegistry::setCursorUp ()

Move cursor key up.

```
262 {
263     _cursorKey = _cursorKey->getParent();
264     return _cursorKey;
265 }
```

8.380.3.17 HxRegKey * HxRegistry::getCursorKey () const [inline]

Get cursor key.

```
121 {
122     return _cursorKey;
123 }
```

8.380.3.18 HxString HxRegistry::getCursorName () const [inline]

Get cursor name.

```
127 {
128     return _cursorName;
129 }
```

8.380.3.19 HxRegKey * HxRegistry::setRootKey () [inline]

Set root key.

```
139 {
140     return _cursorKey = _rootKey;
141 }
```

8.380.3.20 HxRegKey * HxRegistry::getRootKey () const [inline]

Get root key.

```
133 {
134     return _rootKey;
135 }
```

8.380.3.21 STD_OSTREAM & HxRegistry::put (STD_OSTREAM & *os*) const

Put registry on stream.

```
269 {
270     return _rootKey->put(os);
271 }
```

The documentation for this class was generated from the following files:

- **HxRegistry.h**
- HxRegistry.c

8.381 HxRegistryImporter Class Reference

Class for importing stuff into the **HxRegistry** (p. 1088) at program startup via declaration of static variables.

```
#include <HxRegistryImporter.h>
```

Public Methods

- **HxRegistryImporter** (**HxString** fileName)
  Constructor.
- **HxRegistryImporter** (const char *argv[ ])
  Constructor.
- **HxRegistryImporter** (**HxRegistry** &, **HxString** fileName)
  Constructor.
- **HxRegistryImporter** (**HxRegistry** &, const char *argv[ ])
  Constructor.

8.381.1 Detailed Description

Class for importing stuff into the **HxRegistry** (p. 1088) at program startup via declaration of static variables.

8.381.2 Constructor & Destructor Documentation

8.381.2.1 HxRegistryImporter::HxRegistryImporter (HxString *fileName*) [inline]

Constructor.

```
35 {
36     HxRegistry::instance().import(fileName);
37 }
```

8.381.2.2 HxRegistryImporter::HxRegistryImporter (const char * *argv*[ ]) [inline]

Constructor.

```
41 {
42     HxRegistry::instance().import(argv);
43 }
```

8.381.2.3 HxRegistryImporter::HxRegistryImporter (HxRegistry & *registry*, HxString *fileName*) [inline]

Constructor.

```
47 {
48     registry.import(fileName);
49 }
```

8.381.2.4 HxRegistryImporter::HxRegistryImporter (HxRegistry & *registry*, const char * *argv*[ ]) [inline]

Constructor.

```
53 {
54     registry.import(argv);
55 }
```

The documentation for this class was generated from the following file:

- **HxRegistryImporter.h**

8.382 HxRegKey Class Reference

A registry key.

`#include <HxRegKey.h>`

Public Methods

- ~**HxRegKey** ()
  Destructor.
- **HxString getName** () const
  Get name.
- HxRegKey * **getParent** () const
  Get parent.
- HxRegKey * **insertKey** (**HxString** path)
  Insert key.
- void **eraseKey** (**HxString** name)
  Erase key.
- HxRegKey * **findKey** (**HxString** path) const
  Find key.
- HxRegKey * **findKey** (**HxStringListConstIter** first, **HxStringListConstIter** last) const
  Find key.
- int **getKeyList** (**HxRegKeyListBackInserter**) const
  Get list of keys.
- size_t **keyListSize** () const
  Get size of list of keys.
- void **insertValue** (**HxString** name, const **HxRegData** &data)

Insert value.

- void **insertValue** (const **HxRegValue** &value)
  Insert value.
- void **eraseValue** (**HxString** name)
  Erase value.
- const **HxRegValue** * **findValue** (**HxString** name) const
  Find value.
- int **getInt** (**HxString** name) const
  Shorthands for findValue.
- **HxString getString** (**HxString** name) const
  Shorthands for findValue.
- int **getValueList** (**HxRegValueListBackInserter**)
  Shorthands for findValue.
- size_t **valueListSize** () const
  Shorthands for findValue.
- STD_OSTREAM & **put** (STD_OSTREAM &) const
  Put on stream.
- STD_OSTREAM & **put** (STD_OSTREAM &, **HxString** path, int cCode=0) const
  Put on stream.

Static Public Methods

- HxRegKey * **createRootKey** ()
  Create root key.

Friends

- class **HxRegKeyFriend**

8.382.1 Detailed Description

A registry key.

8.382.2 Constructor & Destructor Documentation

8.382.2.1 HxRegKey::~HxRegKey ()

Destructor.

```
47 {
48     if (_keys) {
49         HxRegKeyMap::iterator ptr;
50 
51         for (ptr = _keys->begin(); ptr != _keys->end(); ptr++) {
52             delete (* ptr).second;
53         }
54     }
55 #ifdef HxRegistry_Debug
56     STD_CERR << "HxRegKey::~HxRegKey(), _name = " << _name << STD_ENDL;
57 #endif
58 }
```

8.382.3 Member Function Documentation

8.382.3.1 HxRegKey * HxRegKey::createRootKey () [static]

Create root key.

```
62 {
63     HxRegKey* rootKey = new HxRegKey("", 0);
64     rootKey->_parent = rootKey;
65     return rootKey;
66 }
```

8.382.3.2 HxString HxRegKey::getName () const [inline]

Get name.

```
138 {
139     return _name;
140 }
```

8.382.3.3 HxRegKey * HxRegKey::getParent () const [inline]

Get parent.

```
144 {
145     return _parent;
146 }
```

8.382.3.4 HxRegKey * HxRegKey::insertKey (HxString *path*)

Insert key.

```
93 {
94     HxStringList            nameList;
95     splitString(path, '/', std::back_inserter(nameList));
96     HxStringListConstIter   namePtr = nameList.begin();
97     HxRegKey*               key = this;
98
99     if ((namePtr != nameList.end()) && (*namePtr).empty()) {
100         namePtr++;
101     }
102
103     for (;namePtr != nameList.end(); namePtr++) {
104         if ((*namePtr).empty())
105             continue;
106         key = key->insertOneKey(*namePtr);
107     }
108     return key;
109 }
```

8.382.3.5 void HxRegKey::eraseKey (HxString *name*)

Erase key.

```
113 {
114     HxRegKeyMap::iterator ptr;
115     if (!_keys)
116         return;
117     if ((ptr = _keys->find(name)) != _keys->end()) {
118         HxRegKey* key = (*ptr).second;
119         delete key;
120         _keys->erase(name);
121     }
122 }
```

8.382.3.6 HxRegKey * HxRegKey::findKey (HxString *path*) const

Find key.

```
126 {
127     HxStringList                nameList;
128     splitString(path, '/', std::back_inserter(nameList));
129     return findKey(nameList.begin(), nameList.end());
130 }
```

8.382.3.7 HxRegKey * HxRegKey::findKey (HxStringListConstIter *first*, HxStringListConstIter *last*) const

Find key.

```
160 {
161     HxStringListConstIter       namePtr = first;
162     const HxRegKey*             key = this;
163     HxRegKeyMap::const_iterator ptr;
164
165     for (;namePtr != last; namePtr++) {
166         if ((*namePtr).empty())
```

```
167             continue;
168         if (!key->_keys)
169             return 0;
170         ptr = key->_keys->find(*namePtr);
171         if (ptr == key->_keys->end())
172             return 0;
173         else
174             key = (*ptr).second;
175     }
176     return (HxRegKey*)key;
177 }
```

8.382.3.8 int HxRegKey::getKeyList (HxRegKeyListBackInserter *bi*) const

Get list of keys.

```
181 {
182     HxRegKeyMap::iterator ptr;
183     int                         n = 0;
184 
185     if (!_keys)
186         return 0;
187 
188     for (ptr = _keys->begin(); ptr != _keys->end(); ptr++, n++)
189         *bi++ = (*ptr).second;
190     return n;
191 }
```

8.382.3.9 size_t HxRegKey::keyListSize () const

Get size of list of keys.

```
195 {
196     return _keys->size();
197 }
```

8.382.3.10 void HxRegKey::insertValue (HxString *name*, const HxRegData & *data*) [inline]

Insert value.

```
150 {
151     insertValue(HxRegValue(name, data));
152 }
```

8.382.3.11 void HxRegKey::insertValue (const HxRegValue & *val*)

Insert value.

```
201 {
202     ValueVector::iterator ptr
203         = std::lower_bound(_values.begin(), _values.end(), val);
204     if (ptr == _values.end()) {
```

```
205         _values.insert(ptr, val);
206     } else if ((*ptr).getName() != val.getName()) {
207         _values.insert(ptr, val);
208     } else {
209         (*ptr).setData(val.getData());
210     }
211 }
```

8.382.3.12 void HxRegKey::eraseValue (HxString *name*)

Erase value.

```
215 {
216     ValueVector::iterator ptr = std::lower_bound(
217                                     _values.begin(), _values.end(),
218                                     HxRegValue(name, HxRegData(0)));
219     if ((ptr != _values.end()) && ((*ptr).getName() == name))
220         _values.erase(ptr);
221 }
```

8.382.3.13 const HxRegValue * HxRegKey::findValue (HxString *name*) const

Find value.

```
225 {
226     ValueVector::const_iterator ptr = std::lower_bound(
227                                       _values.begin(), _values.end(),
228                                       HxRegValue(name, HxRegData(0)));
229     if ((ptr != _values.end()) && ((*ptr).getName() == name))
230         return &(*ptr);
231     else
232         return 0;
233 }
```

8.382.3.14 int HxRegKey::getInt (HxString *name*) const [inline]

Shorthands for findValue.

Existence of the values cannot be determined from the return value.

```
156 {
157     const HxRegValue* val = findValue(name);
158     return val ? val->getData().getInt() : 0;
159 }
```

8.382.3.15 HxString HxRegKey::getString (HxString *name*) const [inline]

Shorthands for findValue.

Existence of the values cannot be determined from the return value.

```
163 {
164     const HxRegValue* val = findValue(name);
165     return val ? val->getData().getString() : HxString();
166 }
```

8.382.3.16 int HxRegKey::getValueList (HxRegValueListBackInserter *bi*)

Shorthands for findValue.

Existence of the values cannot be determined from the return value.

```
237 {
238     ValueVector::iterator   ptr;
239     int                     n = 0;
240     for (ptr = _values.begin(); ptr != _values.end(); ptr++, n++)
241         *bi++ = &(*ptr);
242     return n;
243 }
```

8.382.3.17 size_t HxRegKey::valueListSize () const

Shorthands for findValue.

Existence of the values cannot be determined from the return value.

```
247 {
248     return _values.size();
249 }
```

8.382.3.18 STD_OSTREAM & HxRegKey::put (STD_OSTREAM & *os*) const

Put on stream.

```
253 {
254     return put(os, "");
255 }
```

8.382.3.19 STD_OSTREAM & HxRegKey::put (STD_OSTREAM & *os*, HxString *path*, int *cCode* = 0) const

Put on stream.

```
259 {
260     HxRegKeyMap::iterator keyPtr;
261     ValueVector::const_iterator valPtr;
262 
263     if (cCode)
264         os << "\"";
265     os << "[" << path << "/" << _name << "]";
266     if (cCode)
267         os << "\",";
268     os << STD_ENDL;
269 
270     for (valPtr = _values.begin(); valPtr != _values.end(); valPtr++) {
271         if (cCode)
272             os << "\"";
273         (*valPtr).put(os, cCode);
274         if (cCode)
275             os << "\",";
276         os << STD_ENDL;
```

```
277     }
278
279     if (_keys) {
280         if (!_name.empty())
281             path += "/" + _name;
282         for (keyPtr=_keys->begin(); keyPtr != _keys->end(); keyPtr++) {
283             HxRegKey* k = (*keyPtr).second;
284             k->put(os, path, cCode);
285         }
286     }
287     return os;
288 }
```

The documentation for this class was generated from the following files:

- **HxRegKey.h**
- HxRegKey.c

8.383 HxRegKeyList Class Reference

A list of **HxRegKeyPtr** (p. 336)'s, that is pointers to **HxRegKey** (p. 1097)'s.

`#include <HxRegKeyList.h>`

Public Types

- typedef std::back_insert_iterator< HxRegKeyList > **back_insert_iterator**
  back inserter.

8.383.1 Detailed Description

A list of **HxRegKeyPtr** (p. 336)'s, that is pointers to **HxRegKey** (p. 1097)'s.

8.383.2 Member Typedef Documentation

8.383.2.1 typedef std::back_insert_iterator<HxRegKeyList> HxRegKeyList::back_insert_iterator

back inserter.

The documentation for this class was generated from the following file:

- **HxRegKeyList.h**

8.384 HxRegValue Class Reference

A registry value.

`#include <HxRegValue.h>`

Public Methods

- **HxRegValue** ()
  Constructor.
- **HxRegValue** (**HxString** name, const **HxRegData** &data)
  Constructor.
- ~**HxRegValue** ()
  Destructor.
- **HxString getName** () const
  Get name.
- **HxRegData getData** () const
  Get data.
- void **setData** (const **HxRegData** &data)
  Set data.
- int **operator**< (const HxRegValue &rhs) const
  Compare names.
- STD_OSTREAM & **put** (STD_OSTREAM &, int cCode=0) const
  Put on stream.

8.384.1 Detailed Description

A registry value.

8.384.2 Constructor & Destructor Documentation

8.384.2.1 HxRegValue::HxRegValue () [inline]

Constructor.

```
53                          : _data(0)
54 {
55 }
```

8.384.2.2 HxRegValue::HxRegValue (HxString *name*, const HxRegData & *data*) [inline]

Constructor.

```
59     : _name(name), _data(data)
60 {
61 }
```

8.384.2.3 HxRegValue::~HxRegValue () [inline]

Destructor.

```
65 {
66 }
```

8.384.3 Member Function Documentation

8.384.3.1 HxString HxRegValue::getName () const [inline]

Get name.

```
70 {
71     return _name;
72 }
```

8.384.3.2 HxRegData HxRegValue::getData () const [inline]

Get data.

```
76 {
77     return _data;
78 }
```

8.384.3.3 void HxRegValue::setData (const HxRegData & *data*) [inline]

Set data.

```
82 {
83     _data = data;
84 }
```

8.384.3.4 int HxRegValue::operator< (const HxRegValue & *rhs*) const [inline]

Compare names.

```
88 {
89     return _name < rhs._name;
90 }
```

8.384.3.5 STD_OSTREAM & HxRegValue::put (STD_OSTREAM & *os*, int *cCode* = 0) const

Put on stream.

```
16 {
17     HxString quote = cCode ? "\\\"" : "\"";
18     os << quote << _name << quote << "=";
19     _data.put(os, cCode);
20     return os;
21 }
```

The documentation for this class was generated from the following files:

- **HxRegValue.h**
- HxRegValue.c

8.385 HxRegValueList Class Reference

A list of HxRegValuePtr's, that is pointers to **HxRegValue** (p. 1104)'s.

`#include <HxRegValueList.h>`

Public Types

- typedef std::back_insert_iterator< HxRegValueList > **back_insert_iterator**

  back inserter.

8.385.1 Detailed Description

A list of HxRegValuePtr's, that is pointers to **HxRegValue** (p. 1104)'s.

8.385.2 Member Typedef Documentation

8.385.2.1 typedef std::back_insert_iterator<HxRegValueList> HxRegValueList::back_insert_iterator

back inserter.

The documentation for this class was generated from the following file:

- **HxRegValueList.h**

8.386 HxRgbBinary Class Template Reference

Binary display.

`#include <HxRgbBinary.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**

  The "double" arithtype.

Public Methods

- **HxRgbBinary** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "Binary".

8.386.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbBinary< ValT, ValDoubleT >

Binary display.

pixV is 0 or 1.

8.386.2 Member Typedef Documentation

8.386.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbBinary::ArithTypeDouble

The "double" arithtype.

8.386.3 Constructor & Destructor Documentation

8.386.3.1 template<class ValT, class ValDoubleT> HxRgbBinary< ValT, ValDoubleT >::HxRgbBinary (HxTagList &) [inline]

Constructor : empty.

```
32                              {}
```

8.386.4 Member Function Documentation

8.386.4.1 template<class ValT, class ValDoubleT> int HxRgbBinary< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
36                              {
37                                  ValT v = HxScalarInt(0);
38                                  return (pixV == v) ? 0xFF000000 /*black*/
39                                                     : 0xFFFF0000 /*red*/;
40                              }
```

8.386.4.2 template<class ValT, class ValDoubleT> int HxRgbBinary< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
44                              {
45                                  ValDoubleT v = HxScalarInt(0);
46                                  return (pixV == v) ? 0xFF000000 /*black*/
47                                                     : 0xFFFF0000 /*red*/;
48                              }
```

8.386.4.3 template<class ValT, class ValDoubleT> HxString HxRgbBinary< ValT, ValDoubleT >::className () [inline, static]

The name : "Binary".

```
52                              { return HxString("Binary"); }
```

The documentation for this class was generated from the following file:

- **HxRgbBinary.h**

8.387 HxRgbCMY Class Template Reference

CMY display.

`#include <HxRgbCMY.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The "double" arithtype.

Public Methods

- **HxRgbCMY** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "CMY".

8.387.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbCMY< ValT, ValDoubleT >

CMY display.

pixV is CMY color.

8.387.2 Member Typedef Documentation

8.387.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbCMY::ArithTypeDouble

The "double" arithtype.

8.387.3 Constructor & Destructor Documentation

8.387.3.1 template<class ValT, class ValDoubleT> HxRgbCMY< ValT, ValDoubleT >::HxRgbCMY (HxTagList &) [inline]

Constructor : empty.

```
30                              {}
```

8.387.4 Member Function Documentation

8.387.4.1 template<class ValT, class ValDoubleT> int HxRgbCMY< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
34                              {
35                                  return HxColRGB2int(
36                                              HxColCMY2RGB((HxVec3Double) pixV));
37                              }
```

8.387.4.2 template<class ValT, class ValDoubleT> int HxRgbCMY< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
41                              {
42                                  return HxColRGB2int(
43                                              HxColCMY2RGB((HxVec3Double) pixV));
44                              }
```

8.387.4.3 template<class ValT, class ValDoubleT> HxString HxRgbCMY< ValT, ValDoubleT >::className () [inline, static]

The name : ”CMY”.

```
48                              { return HxString("CMY"); }
```

The documentation for this class was generated from the following file:

- **HxRgbCMY.h**

8.388 HxRgbDirect Class Template Reference

Direct display mapping.

`#include <HxRgbDirect.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The ”double” arithtype.

Public Methods

- **HxRgbDirect** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the ”standard” arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the ”double” arithtype.

Static Public Methods

- **HxString className** ()
  The name : ”Direct”.

8.388.1 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.

8.388.2 Member Typedef Documentation

8.388.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbDirect::ArithTypeDouble

The ”double” arithtype.

8.388.3 Constructor & Destructor Documentation

8.388.3.1 template<class ValT, class ValDoubleT> HxRgbDirect< ValT, ValDoubleT >::HxRgbDirect (HxTagList &) [inline]

Constructor : empty.

```
30                              {}
```

8.388.4 Member Function Documentation

8.388.4.1 template<class ValT, class ValDoubleT> int HxRgbDirect< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the ”standard” arithtype.

```
34                              { return HxColRGB2int((HxVec3Int) pixV); }
```

8.388.4.2 template<class ValT, class ValDoubleT> int HxRgbDirect< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the ”double” arithtype.

```
38                              { return HxColRGB2int((HxVec3Double) pixV); }
```

8.388.4.3 template<class ValT, class ValDoubleT> HxString HxRgbDirect< ValT, ValDoubleT >::className () [inline, static]

The name : ”Direct”.

```
42                              { return HxString("Direct"); }
```

The documentation for this class was generated from the following file:

- **HxRgbDirect.h**

8.389 HxRgbDirectNC Class Template Reference

Direct display mapping pixV is RGB value in the range 0-255.

```
#include <HxRgbDirect.h>
```

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The "double" arithtype.

Public Methods

- **HxRgbDirectNC** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "DirectNC".

8.389.1 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.

8.389.2 Member Typedef Documentation

8.389.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbDirectNC::ArithTypeDouble

The "double" arithtype.

8.389.3 Constructor & Destructor Documentation

8.389.3.1 template<class ValT, class ValDoubleT> HxRgbDirectNC< ValT, ValDoubleT >::HxRgbDirectNC (HxTagList &) [inline]

Constructor : empty.

```
58                              {}
```

8.389.4 Member Function Documentation

8.389.4.1 template<class ValT, class ValDoubleT> int HxRgbDirectNC< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
62                              {
63                                  HxVec3Int v = (HxVec3Int) pixV;
64                                  return (255 << 24) | (v.x() << 16) | (v.y() << 8) | v.z();
65                              }
```

8.389.4.2 template<class ValT, class ValDoubleT> int HxRgbDirectNC< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
69                              {
70                                  HxVec3Int v = (HxVec3Int) pixV;
71                                  return (255 << 24) | (v.x() << 16) | (v.y() << 8) | v.z();
72                              }
```

8.389.4.3 template<class ValT, class ValDoubleT> HxString HxRgbDirectNC< ValT, ValDoubleT >::className () [inline, static]

The name : "DirectNC".

```
76                              { return HxString("DirectNC"); }
```

The documentation for this class was generated from the following file:

- **HxRgbDirect.h**

8.390 HxRgbHSI Class Template Reference

HSI display.

`#include <HxRgbHSI.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**

  The "double" arithtype.

Public Methods

- **HxRgbHSI** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "HSI".

8.390.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbHSI< ValT, ValDoubleT >

HSI display.

pixV is HSI color.

8.390.2 Member Typedef Documentation

8.390.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbHSI::ArithTypeDouble

The "double" arithtype.

8.390.3 Constructor & Destructor Documentation

8.390.3.1 template<class ValT, class ValDoubleT> HxRgbHSI< ValT, ValDoubleT >::HxRgbHSI (HxTagList &) [inline]

Constructor : empty.

```
30                              {}
```

8.390.4 Member Function Documentation

8.390.4.1 template<class ValT, class ValDoubleT> int HxRgbHSI< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
34                              {
35                                  return HxColRGB2int(
36                                              HxColHSI2RGB((HxVec3Double) pixV));
37                              }
```

8.390.4.2 template<class ValT, class ValDoubleT> int HxRgbHSI< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
41                              {
42                                  return HxColRGB2int(
43                                              HxColHSI2RGB((HxVec3Double) pixV));
44                              }
```

8.390.4.3 template<class ValT, class ValDoubleT> HxString HxRgbHSI< ValT, ValDoubleT >::className () [inline, static]

The name : "HSI".

```
48                              { return HxString("HSI"); }
```

The documentation for this class was generated from the following file:

- **HxRgbHSI.h**

8.391 HxRgbLab Class Template Reference

Lab display.

`#include <HxRgbLab.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The "double" arithtype.

Public Methods

- **HxRgbLab** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "Lab".

8.391.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLab< ValT, ValDoubleT >

Lab display.

pixV is Lab color.

8.391.2 Member Typedef Documentation

8.391.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbLab::ArithTypeDouble

The "double" arithtype.

8.391.3 Constructor & Destructor Documentation

8.391.3.1 template<class ValT, class ValDoubleT> HxRgbLab< ValT, ValDoubleT >::HxRgbLab (HxTagList &) [inline]

Constructor : empty.

```
30                              {}
```

8.391.4 Member Function Documentation

8.391.4.1 template<class ValT, class ValDoubleT> int HxRgbLab< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
34                              {
35                                  return HxColRGB2int(HxColXYZ2RGB(
36                                              HxColLab2XYZ((HxVec3Double) pixV)));
37                              }
```

8.391.4.2 template<class ValT, class ValDoubleT> int HxRgbLab< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
41                              {
42                                  return HxColRGB2int(HxColXYZ2RGB(
43                                              HxColLab2XYZ((HxVec3Double) pixV)));
44                              }
```

8.391.4.3 template<class ValT, class ValDoubleT> HxString HxRgbLab< ValT, ValDoubleT >::className () [inline, static]

The name : ”Lab”.

```
48                              { return HxString("Lab"); }
```

The documentation for this class was generated from the following file:

- **HxRgbLab.h**

8.392 HxRgbLabel Class Template Reference

Labeled display.

`#include <HxRgbLabel.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The ”double” arithtype.

Public Methods

- **HxRgbLabel** (**HxTagList** &)
  Constructor : creates LUT.
- int **doIt** (const ValT &pixV)
  Actual operation for the ”standard” arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the ”double” arithtype.

Static Public Methods

- **HxString className** ()
  The name : ”Label”.

8.392.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLabel< ValT, ValDoubleT >

Labeled display.

PixV is mapped onto 1 of 8 colors.

8.392.2 Member Typedef Documentation

8.392.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbLabel::ArithTypeDouble

The "double" arithtype.

8.392.3 Constructor & Destructor Documentation

8.392.3.1 template<class ValT, class ValDoubleT> HxRgbLabel< ValT, ValDoubleT >::HxRgbLabel (HxTagList &)

Constructor : creates LUT.

```
58 {
59     colorTable[0] = (255 << 24) | (127 << 16) | (127 << 8) | 127; // 50% grey
60     colorTable[1] = (255 << 24) | (255 << 16) | (  0 << 8) |   0; // red
61     colorTable[2] = (255 << 24) | (  0 << 16) | (255 << 8) |   0; // green
62     colorTable[3] = (255 << 24) | (255 << 16) | (255 << 8) |   0; // yellow
63     colorTable[4] = (255 << 24) | (  0 << 16) | (  0 << 8) | 255; // blue
64     colorTable[5] = (255 << 24) | (255 << 16) | (  0 << 8) | 255; // magenta
65     colorTable[6] = (255 << 24) | (  0 << 16) | (255 << 8) | 255; // cyan
66     colorTable[7] = (255 << 24) | (255 << 16) | (255 << 8) | 255; // white
67     colorTable[8] = (255 << 24) | (  0 << 16) | (  0 << 8) |   0; // black
68 }
```

8.392.4 Member Function Documentation

8.392.4.1 template<class ValT, class ValDoubleT> int HxRgbLabel< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
33                              {
34                                  int pix = (int) pixV.x();
35                                  int mask = (pix == 0) ? 8 : pix % 8;
36                                  return colorTable[mask];
37                              }
```

8.392.4.2 template<class ValT, class ValDoubleT> int HxRgbLabel< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
41                              {
42                                  int pix = (int) pixV.x();
43                                  int mask = (pix == 0) ? 8 : pix % 8;
44                                  return colorTable[mask];
45                              }
```

8.392.4.3 template<class ValT, class ValDoubleT> HxString HxRgbLabel< ValT, ValDoubleT >::className () [inline, static]

The name : ”Label”.

```
49                              { return HxString("Label"); }
```

The documentation for this class was generated from the following file:

- **HxRgbLabel.h**

8.393 HxRgbLogMag Class Template Reference

Log manitude display.

`#include <HxRgbLogMag.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The ”double” arithtype.

Public Methods

- **HxRgbLogMag** (**HxTagList** &tags)
  Constructor.
- int **doIt** (const ValT &pixV)
  Actual operation for the ”standard” arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the ”double” arithtype.

Static Public Methods

- **HxString className** ()
  The name : ”LogMagnitude”.

8.393.1 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.

norm2 already taken in **HxImageRep::getRgbPixels2d** (p. 639)

8.393.2 Member Typedef Documentation

8.393.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbLogMag::ArithTypeDouble

The "double" arithtype.

8.393.3 Constructor & Destructor Documentation

8.393.3.1 template<class ValT, class ValDoubleT> HxRgbLogMag< ValT, ValDoubleT >::HxRgbLogMag (HxTagList & *tags*) [inline]

Constructor.

```
33                              {
34                                  _lowVal = HxGetTag(tags, "lowVal", double(0));
35                                  _highVal = HxGetTag(tags, "highVal", double(255));
36                                  _lowVal -= 1.0;
37                                  _factor = 255.0/::log(_highVal-_lowVal);
38                              }
```

8.393.4 Member Function Documentation

8.393.4.1 template<class ValT, class ValDoubleT> int HxRgbLogMag< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
42                              {
43                                  double v = ::log(pixV.x()-_lowVal);
44                                  int x = (int) (v*_factor);
45                                  return (255 << 24) | (x << 16) | (x << 8) | x;
46                              }
```

8.393.4.2 template<class ValT, class ValDoubleT> int HxRgbLogMag< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
50                              {
51                                  double v = ::log(pixV.x()-_lowVal);
52                                  int x = (int) (v*_factor);
53                                  return (255 << 24) | (x << 16) | (x << 8) | x;
54                              }
```

8.393.4.3 template<class ValT, class ValDoubleT> HxString HxRgbLogMag< ValT, ValDoubleT >::className () [inline, static]

The name : "LogMagnitude".

```
58                              { return HxString("LogMagnitude"); }
```

The documentation for this class was generated from the following file:

- **HxRgbLogMag.h**

8.394 HxRgbLuv Class Template Reference

Luv display.

`#include <HxRgbLuv.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The "double" arithtype.

Public Methods

- **HxRgbLuv** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "Luv".

8.394.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbLuv< ValT, ValDoubleT >

Luv display.

pixV is Luv color.

8.394.2 Member Typedef Documentation

8.394.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbLuv::ArithTypeDouble

The "double" arithtype.

8.394.3 Constructor & Destructor Documentation

8.394.3.1 template<class ValT, class ValDoubleT> HxRgbLuv< ValT, ValDoubleT >::HxRgbLuv (HxTagList &) [inline]

Constructor : empty.

```
30                              {}
```

8.394.4 Member Function Documentation

8.394.4.1 template<class ValT, class ValDoubleT> int HxRgbLuv< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the ”standard” arithtype.

```
34                              {
35                                  return HxColRGB2int(HxColXYZ2RGB(
36                                              HxColLuv2XYZ((HxVec3Double) pixV)));
37                              }
```

8.394.4.2 template<class ValT, class ValDoubleT> int HxRgbLuv< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the ”double” arithtype.

```
41                              {
42                                  return HxColRGB2int(HxColXYZ2RGB(
43                                              HxColLuv2XYZ((HxVec3Double) pixV)));
44                              }
```

8.394.4.3 template<class ValT, class ValDoubleT> HxString HxRgbLuv< ValT, ValDoubleT >::className () [inline, static]

The name : ”Luv”.

```
48                              { return HxString("Luv"); }
```

The documentation for this class was generated from the following file:

- **HxRgbLuv.h**

8.395 HxRgbOOO Class Template Reference

OOO display.

```
#include <HxRgbOOO.h>
```

Public Types

- typedef ValDoubleT **ArithTypeDouble**
  The "double" arithtype.

Public Methods

- **HxRgbOOO** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "OOO".

8.395.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbOOO< ValT, ValDoubleT >

OOO display.

pixV is OOO color.

8.395.2 Member Typedef Documentation

8.395.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbOOO::ArithTypeDouble

The "double" arithtype.

8.395.3 Constructor & Destructor Documentation

8.395.3.1 template<class ValT, class ValDoubleT> HxRgbOOO< ValT, ValDoubleT >::HxRgbOOO (HxTagList &) [inline]

Constructor : empty.

```
31                              {}
```

8.395.4 Member Function Documentation

8.395.4.1 template<class ValT, class ValDoubleT> int HxRgbOOO< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
35                              {
36                                  return HxColRGB2int(
37                                              HxColOOO2RGB((HxVec3Double) pixV));
38                              }
```

8.395.4.2 template<class ValT, class ValDoubleT> int HxRgbOOO< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
42                              {
43                                  return HxColRGB2int(
44                                              HxColOOO2RGB((HxVec3Double) pixV));
45                              }
```

8.395.4.3 template<class ValT, class ValDoubleT> HxString HxRgbOOO< ValT, ValDoubleT >::className () [inline, static]

The name : "OOO".

```
49                              { return HxString("OOO"); }
```

The documentation for this class was generated from the following file:

- **HxRgbOOO.h**

8.396 HxRgbStretch Class Template Reference

Stretched display.

`#include <HxRgbStretch.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**

  The "double" arithtype.

Public Methods

- **HxRgbStretch** (**HxTagList** &tags)
  Constructor.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "Stretch".

8.396.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbStretch< ValT, ValDoubleT >

Stretched display.

pixV is strectched between lowVal and highVal.

8.396.2 Member Typedef Documentation

8.396.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbStretch::ArithTypeDouble

The "double" arithtype.

8.396.3 Constructor & Destructor Documentation

8.396.3.1 template<class ValT, class ValDoubleT> HxRgbStretch< ValT, ValDoubleT >::HxRgbStretch (HxTagList & *tags*) [inline]

Constructor.

```
30                              {
31                                  _lowVal = HxGetTag(tags, "lowVal", double(0));
32                                  _highVal = HxGetTag(tags, "highVal", double(255));
33                                  _interval = _highVal - _lowVal;
34                              }
```

8.396.4 Member Function Documentation

8.396.4.1 template<class ValT, class ValDoubleT> int HxRgbStretch< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
38                              {
39                                  HxVec3Double v = (HxVec3Double) pixV;
40                                  int x = (int) (((v.x() - _lowVal) / _interval) * 255);
41                                  int y = (int) (((v.y() - _lowVal) / _interval) * 255);
42                                  int z = (int) (((v.z() - _lowVal) / _interval) * 255);
43                                  return (255 << 24) | (x << 16) | (y << 8) | z;
44                              }
```

8.396.4.2 template<class ValT, class ValDoubleT> int HxRgbStretch< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
48                              {
49                                  HxVec3Double v = (HxVec3Double) pixV;
50                                  int x = (int) (((v.x() - _lowVal) / _interval) * 255);
51                                  int y = (int) (((v.y() - _lowVal) / _interval) * 255);
52                                  int z = (int) (((v.z() - _lowVal) / _interval) * 255);
53                                  return (255 << 24) | (x << 16) | (y << 8) | z;
54                              }
```

8.396.4.3 template<class ValT, class ValDoubleT> HxString HxRgbStretch< ValT, ValDoubleT >::className () [inline, static]

The name : "Stretch".

```
58                              { return HxString("Stretch"); }
```

The documentation for this class was generated from the following file:

- **HxRgbStretch.h**

8.397 HxRgbXYZ Class Template Reference

XYZ display.

`#include <HxRgbXYZ.h>`

Public Types

- typedef ValDoubleT **ArithTypeDouble**

  The "double" arithtype.

Public Methods

- **HxRgbXYZ** (**HxTagList** &)
  Constructor : empty.
- int **doIt** (const ValT &pixV)
  Actual operation for the "standard" arithtype.
- int **doItDouble** (const ValDoubleT &pixV)
  Actual operation for the "double" arithtype.

Static Public Methods

- **HxString className** ()
  The name : "XYZ".

8.397.1 Detailed Description

template<class ValT, class ValDoubleT> class HxRgbXYZ< ValT, ValDoubleT >

XYZ display.

pixV is XYZ color.

8.397.2 Member Typedef Documentation

8.397.2.1 template<class ValT, class ValDoubleT> typedef ValDoubleT HxRgbXYZ::ArithTypeDouble

The "double" arithtype.

8.397.3 Constructor & Destructor Documentation

8.397.3.1 template<class ValT, class ValDoubleT> HxRgbXYZ< ValT, ValDoubleT >::HxRgbXYZ (HxTagList &) [inline]

Constructor : empty.

```
30                              {}
```

8.397.4 Member Function Documentation

8.397.4.1 template<class ValT, class ValDoubleT> int HxRgbXYZ< ValT, ValDoubleT >::doIt (const ValT & *pixV*) [inline]

Actual operation for the "standard" arithtype.

```
34                              {
35                                  return HxColRGB2int(
36                                              HxColXYZ2RGB((HxVec3Double) pixV));
37                              }
```

8.397.4.2 template<class ValT, class ValDoubleT> int HxRgbXYZ< ValT, ValDoubleT >::doItDouble (const ValDoubleT & *pixV*) [inline]

Actual operation for the "double" arithtype.

```
41                              {
42                                  return HxColRGB2int(
43                                              HxColXYZ2RGB((HxVec3Double) pixV));
44                              }
```

8.397.4.3 template<class ValT, class ValDoubleT> HxString HxRgbXYZ< ValT, ValDoubleT >::className () [inline, static]

The name : "XYZ".

```
48                              { return HxString("XYZ"); }
```

The documentation for this class was generated from the following file:

- **HxRgbXYZ.h**

8.398 HxSampledBSplineCurve Class Reference

Class definition for sampled BSpline curves.

`#include <HxSampledBSplineCurve.h>`

Public Methods

- **HxSampledBSplineCurve** ()

  Construct default curve.

- **HxSampledBSplineCurve** (const **HxBSplineCurve** &curve, int nSamples, HxBSplineSampling-Alg 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 **HxSampledBSplineInterval** &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.

8.398.1 Detailed Description

Class definition for sampled BSpline curves.

8.398.2 Constructor & Destructor Documentation

8.398.2.1 HxSampledBSplineCurve::HxSampledBSplineCurve ()

Construct default curve.

```
19 {
20     _ident = _nr++;
21     makeDefault();
22 }
```

8.398.2.2 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 }
```

8.398.2.3 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 }
```

8.398.2.4 HxSampledBSplineCurve::~HxSampledBSplineCurve ()

Destructor.

```
73 {
74 }
```

8.398.3 Member Function Documentation

8.398.3.1 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 }
```

8.398.3.2 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 }
```

8.398.3.3 int HxSampledBSplineCurve::ident () const [inline]

Get the identifier.

```
219 {
220     return _ident;
221 }
```

8.398.3.4 HxBSplineCurve HxSampledBSplineCurve::continuousCurve () const [inline]

Get the continuous curve of this object.

```
225 {
226     return _curve;
227 }
```

8.398.3.5 HxBSplineType HxSampledBSplineCurve::curveType () const [inline]

Get the type of this curve.

```
231 {
232     return _curve.curveType();
233 }
```

8.398.3.6 HxBSplineSamplingAlg HxSampledBSplineCurve::samplingAlg () const [inline]

Get the sampling algorithm.

```
255 {
256     return _samplingAlg;
257 }
```

8.398.3.7 int HxSampledBSplineCurve::nSamples () const [inline]

Get the number of samples.

```
261 {
262     return _t.size();
263 }
```

8.398.3.8 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 }
```

8.398.3.9 vector< double > HxSampledBSplineCurve::allSampledT () const [inline]

Get value of t for all samples.

```
267 {
268     return _t;
269 }
```

8.398.3.10 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 }
```

8.398.3.11 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 }
```

8.398.3.12 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 }
```

8.398.3.13 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 }
```

8.398.3.14 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 }
```

8.398.3.15 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 }
```

8.398.3.16 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 }
```

8.398.3.17 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 }
```

8.398.3.18 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 }
```

8.398.3.19 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 }
```

8.398.3.20 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 }
```

8.398.3.21 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 }
```

8.398.3.22 HxPolyline2d HxSampledBSplineCurve::CPoly () const [inline]

Polyline with all sampled curve points.

```
327 {
328     return HxPolyline2d(AllC(), (curveType() == closed));
329 }
```

8.398.3.23 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 }
```

8.398.3.24 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 }
```

8.398.3.25 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 }
```

8.398.3.26 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 }
```

8.398.3.27 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 }
```

8.398.3.28 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 }
```

8.398.3.29 double HxSampledBSplineCurve::length () const [inline]

total curve length.

```
363 {
364     return _curve.length(nSamples());
365 }
```

8.398.3.30 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 }
```

8.398.3.31 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 }
```

8.398.3.32 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 }
```

8.398.3.33 int HxSampledBSplineCurve::numP () const [inline]

Get number of control points.

```
237 {
238     return _curve.numP();
239 }
```

8.398.3.34 HxPointSetR2 HxSampledBSplineCurve::allP () const [inline]

Get all control points.

```
243 {
244     return _curve.allP();
245 }
```

8.398.3.35 HxPolyline2d HxSampledBSplineCurve::controlP () const [inline]

Get the control polygon.

```
249 {
250     return HxPolyline2d(allP(), (curveType() == closed));
251 }
```

8.398.3.36 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 }
```

8.398.3.37 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 }
```

8.398.3.38 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

8.399 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.

8.399.1 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.

8.399.2 Constructor & Destructor Documentation

8.399.2.1 HxSampledBSplineInterval::HxSampledBSplineInterval () [inline]

Default constructor.

```
71                                                  : pair<int,int>()
72 {
73     _max = 0;
74     _wrap = 0;
75 }
```

8.399.2.2 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 }
```

8.399.2.3 HxSampledBSplineInterval::~HxSampledBSplineInterval () [inline]

Destructor.

```
87 {
88 }
```

8.399.3 Member Function Documentation

8.399.3.1 int HxSampledBSplineInterval::begin () const [inline]

First sample.

```
92 {
93     return first;
94 }
```

8.399.3.2 int HxSampledBSplineInterval::end () const [inline]

Last sample.

```
98 {
99     return second;
100 }
```

8.399.3.3 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 }
```

8.399.3.4 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 }
```

8.399.3.5 int HxSampledBSplineInterval::middle () const [inline]

central sample in interval.

```
126 {
127     return ratio(0.5);
128 }
```

8.399.3.6 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 }
```

8.399.3.7 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**

8.400 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. 1164).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 1262).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.400.1 Detailed Description

Class definition scalar double.

8.400.2 Constructor & Destructor Documentation

8.400.2.1 HxScalarDouble::HxScalarDouble () [inline]

Default constructor.

```
320 {
321 }
```

8.400.2.2 HxScalarDouble::HxScalarDouble (double *v*) [inline]

Conversion from native type.

```
325     : _value(v)
326 {
327 }
```

8.400.2.3 HxScalarDouble::HxScalarDouble (const HxScalarDouble & *rhs*) [inline]

Copy constructor.

```
331     : _value(rhs._value)
332 {
333 }
```

8.400.3 Member Function Documentation

8.400.3.1 int HxScalarDouble::dim () const [inline]

Dimensionality.

```
357 {
358     return 1;
359 }
```

8.400.3.2 double HxScalarDouble::x () const [inline]

Value of (first) element.

```
363 {
364     return _value;
365 }
```

8.400.3.3 double HxScalarDouble::getValue (int *dimension*) const [inline]

Element in given dimension.

```
369 {
370     return _value;
371 }
```

8.400.3.4 HxScalarDouble::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 1164).

```
26 {
27     return int(_value);
28 }
```

8.400.3.5 HxScalarDouble::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 1281).

```
31 {
32     return HxVec2Int(int(_value), int(_value));
33 }
```

8.400.3.6 HxScalarDouble::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 1262).

```
36 {
37     return HxVec2Double(_value, _value);
38 }
```

8.400.3.7 HxScalarDouble::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 1321).

```
41 {
42     return HxVec3Int(int(_value), int(_value), int(_value));
43 }
```

8.400.3.8 HxScalarDouble::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 1301).

```
46 {
47     return HxVec3Double(_value, _value, _value);
48 }
```

8.400.3.9 HxScalarDouble::operator HxComplex () const

Cast to **HxComplex** (p. 506).

```
51 {
52     return HxComplex(_value, 0.0);
53 }
```

8.400.3.10 int HxScalarDouble::operator== (const HxScalarDouble & *v*) const [inline]

Equal.

```
387 {
388     return (_value == v._value);
389 }
```

8.400.3.11 int HxScalarDouble::operator!= (const HxScalarDouble & *v*) const [inline]

Not equal.

```
393 {
394     return (_value != v._value);
395 }
```

8.400.3.12 int HxScalarDouble::operator< (const HxScalarDouble & *v*) const [inline]

Less than.

```
399 {
400     return (_value < v._value);
401 }
```

8.400.3.13 int HxScalarDouble::operator<= (const HxScalarDouble & *v*) const [inline]

Less equal.

```
405 {
406     return (_value <= v._value);
407 }
```

8.400.3.14 int HxScalarDouble::operator> (const HxScalarDouble & *v*) const [inline]

Greater than.

```
411 {
412     return (_value > v._value);
413 }
```

8.400.3.15 int HxScalarDouble::operator>= (const HxScalarDouble & *v*) const [inline]

Greater equal.

```
417 {
418     return (_value >= v._value);
419 }
```

8.400.3.16 HxScalarDouble HxScalarDouble::operator- () const [inline]

Negation.

```
423 {
424     return HxScalarDouble(-_value);
425 }
```

8.400.3.17 HxScalarDouble HxScalarDouble::complement () const [inline]

Complement.

```
429 {
430     return HxScalarDouble(-_value);
431 }
```

8.400.3.18 HxScalarDouble HxScalarDouble::abs () const [inline]

Absolute value.

```
435 {
436     return HxScalarDouble(fabs(_value));
437 }
```

8.400.3.19 HxScalarDouble HxScalarDouble::ceil () const [inline]

Ceiling.

```
441 {
442     return HxScalarDouble(::ceil(_value));
443 }
```

8.400.3.20 HxScalarDouble HxScalarDouble::floor () const [inline]

Floor.

```
447 {
448     return HxScalarDouble(::floor(_value));
449 }
```

8.400.3.21 HxScalarDouble HxScalarDouble::round () const [inline]

Round.

```
453 {
454     return HxScalarDouble((int) (_value + ((_value >= 0) ? 0.5 : -0.5)));
455 }
```

8.400.3.22 HxScalarDouble HxScalarDouble::sum () const [inline]

Sum.

```
459 {
460     return *this;
461 }
```

8.400.3.23 HxScalarDouble HxScalarDouble::product () const [inline]

Product.

```
465 {
466     return *this;
467 }
```

8.400.3.24 HxScalarDouble HxScalarDouble::min () const [inline]

Minimum.

```
471 {
472     return *this;
473 }
```

8.400.3.25 HxScalarDouble HxScalarDouble::max () const [inline]

Maximum.

```
477 {
478     return *this;
479 }
```

8.400.3.26 HxScalarDouble HxScalarDouble::norm1 () const [inline]

L1 norm.

```
483 {
484     return HxScalarDouble(fabs(_value));
485 }
```

8.400.3.27 HxScalarDouble HxScalarDouble::norm2 () const

L2 norm.

```
57 {
58     return HxScalarDouble(fabs(_value));
59 }
```

8.400.3.28 HxScalarDouble HxScalarDouble::normInf () const [inline]

L infinity norm.

```
489 {
490     return HxScalarDouble(fabs(_value));
491 }
```

8.400.3.29 HxScalarDouble HxScalarDouble::sqrt () const [inline]

Square root.

```
495 {
496     return HxScalarDouble(::sqrt(_value));
497 }
```

8.400.3.30 HxScalarDouble HxScalarDouble::sin () const [inline]

Sine.

```
501 {
502     return HxScalarDouble(::sin(_value));
503 }
```

8.400.3.31 HxScalarDouble HxScalarDouble::cos () const [inline]

Cosine.

```
507 {
508     return HxScalarDouble(::cos(_value));
509 }
```

8.400.3.32 HxScalarDouble HxScalarDouble::tan () const [inline]

Tangent.

```
513 {
514     return HxScalarDouble(::tan(_value));
515 }
```

8.400.3.33 HxScalarDouble HxScalarDouble::asin () const [inline]

Arc sine.

```
519 {
520     return HxScalarDouble(::asin(_value));
521 }
```

8.400.3.34 HxScalarDouble HxScalarDouble::acos () const [inline]

Arc cosine.

```
525 {
526     return HxScalarDouble(::acos(_value));
527 }
```

8.400.3.35 HxScalarDouble HxScalarDouble::atan () const [inline]

Arc tangent.

```
531 {
532     return HxScalarDouble(::atan(_value));
533 }
```

8.400.3.36 HxScalarDouble HxScalarDouble::atan2 () const

Arc tangent.

```
63 {
64     return HxScalarDouble(::atan(_value));
65 }
```

8.400.3.37 HxScalarDouble HxScalarDouble::sinh () const [inline]

Hyperbolic sine.

```
537 {
538     return HxScalarDouble(::sinh(_value));
539 }
```

8.400.3.38 HxScalarDouble HxScalarDouble::cosh () const [inline]

Hyperbolic cosine.

```
543 {
544     return HxScalarDouble(::cosh(_value));
545 }
```

8.400.3.39 HxScalarDouble HxScalarDouble::tanh () const [inline]

Hyperbolic tangent.

```
549 {
550     return HxScalarDouble(::tanh(_value));
551 }
```

8.400.3.40 HxScalarDouble HxScalarDouble::exp () const [inline]

Exponent.

```
555 {
556     return HxScalarDouble(::exp(_value));
557 }
```

8.400.3.41 HxScalarDouble HxScalarDouble::log () const [inline]

Natural logarithm.

```
561 {
562     return HxScalarDouble(::log(_value));
563 }
```

8.400.3.42 HxScalarDouble HxScalarDouble::log10 () const [inline]

Base 10 logarithm.

```
567 {
568     return HxScalarDouble(::log10(_value));
569 }
```

8.400.3.43 HxScalarDouble & HxScalarDouble::operator+= (const HxScalarDouble & *v*) [inline]

Addition and assignment.

```
573 {
574     _value += v._value;
575     return *this;
576 }
```

8.400.3.44 HxScalarDouble & HxScalarDouble::operator-= (const HxScalarDouble & *v*) [inline]

Subtraction and assignment.

```
580 {
581     _value -= v._value;
582     return *this;
583 }
```

8.400.3.45 HxScalarDouble & HxScalarDouble::operator *= (const HxScalarDouble & *v*) [inline]

Multiplication and assignment.

```
587 {
588     _value *= v._value;
589     return *this;
590 }
```

8.400.3.46 HxScalarDouble & HxScalarDouble::operator/= (const HxScalarDouble & *v*) [inline]

Division and assignment.

```
594 {
595     _value /= v._value;
596     return *this;
597 }
```

8.400.3.47 HxScalarDouble HxScalarDouble::min (const HxScalarDouble & *v*) const [inline]

Minimum.

```
625 {
626     return (operator<(v)) ? (*this) : v;
627 }
```

8.400.3.48 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 }
```

8.400.3.49 HxScalarDouble HxScalarDouble::max (const HxScalarDouble & *v*) const [inline]

Maximum.

```
640 {
641     return (operator>(v)) ? (*this) : v;
642 }
```

8.400.3.50 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 }
```

8.400.3.51 HxScalarDouble HxScalarDouble::inf (const HxScalarDouble & *v*) const [inline]

Infimum.

```
655 {
656     return (operator<(v)) ? (*this) : v;
657 }
```

8.400.3.52 HxScalarDouble & HxScalarDouble::infAssign (const HxScalarDouble & *v*) [inline]

Infimum and assignment.

```
661 {
662     _value = (_value < v._value) ? _value : v._value;
663     return *this;
664 }
```

8.400.3.53 HxScalarDouble HxScalarDouble::sup (const HxScalarDouble & *v*) const [inline]

Supremum.

```
668 {
669     return (operator>(v)) ? (*this) : v;
670 }
```

8.400.3.54 HxScalarDouble & HxScalarDouble::supAssign (const HxScalarDouble & *v*) [inline]

Supremum and assignment.

```
674 {
675     _value = (_value > v._value) ? _value : v._value;
676     return *this;
677 }
```

8.400.3.55 HxScalarDouble HxScalarDouble::pow (const HxScalarDouble & *v*) const [inline]

Power.

```
681 {
682     return HxScalarDouble(::pow(_value, v._value));
683 }
```

8.400.3.56 HxScalarDouble HxScalarDouble::mod (const HxScalarDouble & *v*) const [inline]

Modulo.

```
687 {
688     return (*this);
689 }
```

8.400.3.57 HxScalarDouble HxScalarDouble::and (const HxScalarDouble & *v*) const [inline]

And.

```
693 {
694     return (*this);
695 }
```

8.400.3.58 HxScalarDouble HxScalarDouble::or (const HxScalarDouble & *v*) const [inline]

Or.

```
699 {
700     return (*this);
701 }
```

8.400.3.59 HxScalarDouble HxScalarDouble::xor (const HxScalarDouble & *v*) const [inline]

Xor.

```
705 {
706     return (*this);
707 }
```

8.400.3.60 HxScalarDouble HxScalarDouble::leftShift (const HxScalarDouble & *v*) const [inline]

Left shift.

```
711 {
712     return (*this);
713 }
```

8.400.3.61 HxScalarDouble HxScalarDouble::rightShift (const HxScalarDouble & *v*) const [inline]

Right shift.

```
717 {
718     return (*this);
719 }
```

8.400.3.62 HxScalarDouble HxScalarDouble::dot (const HxScalarDouble & *v*) const

Dot product.

```
69 {
70     return _value * v._value;
71 }
```

8.400.3.63 HxScalarDouble HxScalarDouble::cross (const HxScalarDouble & *v*) const [inline]

Cross product.

```
723 {
724     return HxScalarDouble(0.0);
725 }
```

8.400.3.64 STD_OSTREAM & HxScalarDouble::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
75 {
76     return os << _value;
77 }
```

8.400.3.65 HxString HxScalarDouble::toString () const [inline]

Value as a string.

```
729 {
730     return makeString(_value);
731 }
```

8.400.4 Friends And Related Function Documentation

8.400.4.1 HxScalarDouble operator+ (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Addition.

```
601 {
602     return HxScalarDouble(v1._value + v2._value);
603 }
```

8.400.4.2 HxScalarDouble operator- (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Subtraction.

```
607 {
608     return HxScalarDouble(v1._value - v2._value);
609 }
```

8.400.4.3 HxScalarDouble operator * (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Multiplication.

```
613 {
614     return HxScalarDouble(v1._value * v2._value);
615 }
```

8.400.4.4 HxScalarDouble operator/ (const HxScalarDouble & *v1*, const HxScalarDouble & *v2*) [friend]

Division.

```
619 {
620     return HxScalarDouble(v1._value / v2._value);
621 }
```

8.400.5 Member Data Documentation

8.400.5.1 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.

8.400.5.2 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

8.401 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. 1145).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 1262).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.401.1 Detailed Description

Class definition scalar integer.

8.401.2 Constructor & Destructor Documentation

8.401.2.1 HxScalarInt::HxScalarInt () [inline]

Default constructor.

```
317 {
318 }
```

8.401.2.2 HxScalarInt::HxScalarInt (int *v*) [inline]

Conversion from native type.

```
322 {
323     _value = v;
324 }
```

8.401.2.3 HxScalarInt::HxScalarInt (const HxScalarInt & *v*) [inline]

Copy constructor.

```
328 {
329     _value = v._value;
330 }
```

8.401.3 Member Function Documentation

8.401.3.1 int HxScalarInt::dim () const [inline]

Dimensionality.

```
340 {
341     return 1;
342 }
```

8.401.3.2 int HxScalarInt::x () const [inline]

Value of (first) element.

```
346 {
347     return _value;
348 }
```

8.401.3.3 int HxScalarInt::getValue (int *dimension*) const [inline]

Element in given dimension.

```
352 {
353     return _value;
354 }
```

8.401.3.4 HxScalarInt::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 1145).

```
26 {
27     return HxScalarDouble(_value);
28 }
```

8.401.3.5 HxScalarInt::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 1281).

```
31 {
32     return HxVec2Int(_value, _value);
33 }
```

8.401.3.6 HxScalarInt::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 1262).

```
36 {
37     return HxVec2Double(_value, _value);
38 }
```

8.401.3.7 HxScalarInt::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 1321).

```
41 {
42     return HxVec3Int(_value, _value, _value);
43 }
```

8.401.3.8 HxScalarInt::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 1301).

```
46 {
47     return HxVec3Double(_value, _value, _value);
48 }
```

8.401.3.9 HxScalarInt::operator HxComplex () const

Cast to **HxComplex** (p. 506).

```
51 {
52     return HxComplex(_value, 0.0);
53 }
```

8.401.3.10 int HxScalarInt::operator== (const HxScalarInt & *v*) const [inline]

Equal.

```
370 {
371     return (_value == v._value);
372 }
```

8.401.3.11 int HxScalarInt::operator!= (const HxScalarInt & *v*) const [inline]

Not equal.

```
376 {
377     return (_value != v._value);
378 }
```

8.401.3.12 int HxScalarInt::operator< (const HxScalarInt & *v*) const [inline]

Less than.

```
382 {
383     return (_value < v._value);
384 }
```

8.401.3.13 int HxScalarInt::operator<= (const HxScalarInt & *v*) const [inline]

Less equal.

```
388 {
389     return (_value <= v._value);
390 }
```

8.401.3.14 int HxScalarInt::operator> (const HxScalarInt & *v*) const [inline]

Greater than.

```
394 {
395     return (_value > v._value);
396 }
```

8.401.3.15 int HxScalarInt::operator>= (const HxScalarInt & *v*) const [inline]

Greater equal.

```
400 {
401     return (_value >= v._value);
402 }
```

8.401.3.16 HxScalarInt HxScalarInt::operator- () const [inline]

Negation.

```
406 {
407     return HxScalarInt(-_value);
408 }
```

8.401.3.17 HxScalarInt HxScalarInt::complement () const [inline]

Complement.

```
412 {
413     return HxScalarInt(~_value);
414 }
```

8.401.3.18 HxScalarInt HxScalarInt::abs () const [inline]

Absolute value.

```
418 {
419     return HxScalarInt(::abs(_value));
420 }
```

8.401.3.19 HxScalarInt HxScalarInt::ceil () const [inline]

Ceiling.

```
424 {
425     return *this;
426 }
```

8.401.3.20 HxScalarInt HxScalarInt::floor () const [inline]

Floor.

```
430 {
431     return *this;
432 }
```

8.401.3.21 HxScalarInt HxScalarInt::round () const [inline]

Round.

```
436 {
437     return *this;
438 }
```

8.401.3.22 HxScalarInt HxScalarInt::sum () const [inline]

Sum.

```
442 {
443     return *this;
444 }
```

8.401.3.23 HxScalarInt HxScalarInt::product () const [inline]

Product.

```
448 {
449     return *this;
450 }
```

8.401.3.24 HxScalarInt HxScalarInt::min () const [inline]

Minimum.

```
454 {
455     return *this;
456 }
```

8.401.3.25 HxScalarInt HxScalarInt::max () const [inline]

Maximum.

```
460 {
461     return *this;
462 }
```

8.401.3.26 HxScalarInt HxScalarInt::norm1 () const [inline]

L1 norm.

```
466 {
467     return HxScalarInt(::abs(_value));
468 }
```

8.401.3.27 HxScalarDouble HxScalarInt::norm2 () const

L2 norm.

```
57 {
58     return HxScalarDouble(::abs(_value));
59 }
```

8.401.3.28 HxScalarInt HxScalarInt::normInf () const [inline]

L infinity norm.

```
472 {
473     return HxScalarInt(::abs(_value));
474 }
```

8.401.3.29 HxScalarDouble HxScalarInt::sqrt () const

Square root.

```
63 {
64     return HxScalarDouble(::sqrt(double(_value)));
65 }
```

8.401.3.30 HxScalarDouble HxScalarInt::sin () const

Sine.

```
69 {
70     return HxScalarDouble(::sin(double(_value)));
71 }
```

8.401.3.31 HxScalarDouble HxScalarInt::cos () const

Cosine.

```
75 {
76     return HxScalarDouble(::cos(double(_value)));
77 }
```

8.401.3.32 HxScalarDouble HxScalarInt::tan () const

Tangent.

```
81 {
82     return HxScalarDouble(::tan(double(_value)));
83 }
```

8.401.3.33 HxScalarDouble HxScalarInt::asin () const

Arc sine.

```
87 {
88     return HxScalarDouble(::asin(double(_value)));
89 }
```

8.401.3.34 HxScalarDouble HxScalarInt::acos () const

Arc cosine.

```
93 {
94     return HxScalarDouble(::acos(double(_value)));
95 }
```

8.401.3.35 HxScalarDouble HxScalarInt::atan () const

Arc tangent.

```
99 {
100     return HxScalarDouble(::atan(double(_value)));
101 }
```

8.401.3.36 HxScalarDouble HxScalarInt::atan2 () const

Arc tangent.

```
105 {
106     return HxScalarDouble(::atan(double(_value)));
107 }
```

8.401.3.37 HxScalarDouble HxScalarInt::sinh () const

Hyperbolic sine.

```
111 {
112     return HxScalarDouble(::sinh(double(_value)));
113 }
```

8.401.3.38 HxScalarDouble HxScalarInt::cosh () const

Hyperbolic cosine.

```
117 {
118     return HxScalarDouble(::cosh(double(_value)));
119 }
```

8.401.3.39 HxScalarDouble HxScalarInt::tanh () const

Hyperbolic tangent.

```
123 {
124     return HxScalarDouble(::tanh(double(_value)));
125 }
```

8.401.3.40 HxScalarDouble HxScalarInt::exp () const

Exponent.

```
129 {
130     return HxScalarDouble(::exp(double(_value)));
131 }
```

8.401.3.41 HxScalarDouble HxScalarInt::log () const

Natural logarithm.

```
135 {
136     return HxScalarDouble(::log(double(_value)));
137 }
```

8.401.3.42 HxScalarDouble HxScalarInt::log10 () const

Base 10 logarithm.

```
141 {
142     return HxScalarDouble(::log10(double(_value)));
143 }
```

8.401.3.43 HxScalarInt & HxScalarInt::operator+= (const HxScalarInt & *v*) [inline]

Addition and assignment.

```
478 {
479     _value += v._value;
480     return *this;
481 }
```

8.401.3.44 HxScalarInt & HxScalarInt::operator-= (const HxScalarInt & *v*) [inline]

Subtraction and assignment.

```
485 {
486     _value -= v._value;
487     return *this;
488 }
```

8.401.3.45 HxScalarInt & HxScalarInt::operator *= (const HxScalarInt & *v*) [inline]

Multiplication and assignment.

```
492 {
493     _value *= v._value;
494     return *this;
495 }
```

8.401.3.46 HxScalarInt & HxScalarInt::operator/= (const HxScalarInt & *v*) [inline]

Division and assignment.

```
499 {
500     _value /= v._value;
501     return *this;
502 }
```

8.401.3.47 HxScalarInt HxScalarInt::min (const HxScalarInt & *v*) const [inline]

Minimum.

```
530 {
531     return (operator<(v)) ? (*this) : v;
532 }
```

8.401.3.48 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 }
```

8.401.3.49 HxScalarInt HxScalarInt::max (const HxScalarInt & *v*) const [inline]

Maximum.

```
545 {
546     return (operator>(v)) ? (*this) : v;
547 }
```

8.401.3.50 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 }
```

8.401.3.51 HxScalarInt HxScalarInt::inf (const HxScalarInt & *v*) const [inline]

Infimum.

```
560 {
561     return (operator<(v)) ? (*this) : v;
562 }
```

8.401.3.52 HxScalarInt & HxScalarInt::infAssign (const HxScalarInt & *v*) [inline]

Infimum and assignment.

```
566 {
567     _value = (_value < v._value) ? _value : v._value;
568     return *this;
569 }
```

8.401.3.53 HxScalarInt HxScalarInt::sup (const HxScalarInt & *v*) const [inline]

Supremum.

```
573 {
574     return (operator>(v)) ? (*this) : v;
575 }
```

8.401.3.54 HxScalarInt & HxScalarInt::supAssign (const HxScalarInt & *v*) [inline]

Supremum and assignment.

```
579 {
580     _value = (_value > v._value) ? _value : v._value;
581     return *this;
582 }
```

8.401.3.55 HxScalarInt HxScalarInt::pow (const HxScalarInt & *v*) const [inline]

Power.

```
586 {
587     return HxScalarInt((int) (::pow(_value, v._value) + 0.5));
588 }
```

8.401.3.56 HxScalarInt HxScalarInt::mod (const HxScalarInt & *v*) const [inline]

Modulo.

```
592 {
593     return HxScalarInt(_value % v._value);
594 }
```

8.401.3.57 HxScalarInt HxScalarInt::and (const HxScalarInt & *v*) const [inline]

And.

```
598 {
599     return HxScalarInt(_value & v._value);
600 }
```

8.401.3.58 HxScalarInt HxScalarInt::or (const HxScalarInt & *v*) const [inline]

Or.

```
604 {
605     return HxScalarInt(_value | v._value);
606 }
```

8.401.3.59 HxScalarInt HxScalarInt::xor (const HxScalarInt & *v*) const [inline]

Xor.

```
610 {
611     return HxScalarInt(_value ^ v._value);
612 }
```

8.401.3.60 HxScalarInt HxScalarInt::leftShift (const HxScalarInt & *v*) const [inline]

Left shift.

```
616 {
617     return HxScalarInt(_value << v._value);
618 }
```

8.401.3.61 HxScalarInt HxScalarInt::rightShift (const HxScalarInt & *v*) const [inline]

Right shift.

```
622 {
623     return HxScalarInt(_value >> v._value);
624 }
```

8.401.3.62 HxScalarInt HxScalarInt::dot (const HxScalarInt & *v*) const

Dot product.

```
147 {
148     return _value * v._value;
149 }
```

8.401.3.63 HxScalarInt HxScalarInt::cross (const HxScalarInt & *v*) const [inline]

Cross product.

```
628 {
629     return 0;
630 }
```

8.401.3.64 STD_OSTREAM & HxScalarInt::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
153 {
154     return os << _value;
155 }
```

8.401.3.65 HxString HxScalarInt::toString () const [inline]

Value as a string.

```
634 {
635     return makeString(_value);
636 }
```

8.401.4 Friends And Related Function Documentation

8.401.4.1 HxScalarInt operator+ (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Addition.

```
506 {
507     return HxScalarInt(v1._value + v2._value);
508 }
```

8.401.4.2 HxScalarInt operator- (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Subtraction.

```
512 {
513     return HxScalarInt(v1._value - v2._value);
514 }
```

8.401.4.3 HxScalarInt operator * (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Multiplication.

```
518 {
519     return HxScalarInt(v1._value * v2._value);
520 }
```

8.401.4.4 HxScalarInt operator/ (const HxScalarInt & *v1*, const HxScalarInt & *v2*) [friend]

Division.

```
524 {
525     return HxScalarInt(v1._value / v2._value);
526 }
```

8.401.5 Member Data Documentation

8.401.5.1 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.

8.401.5.2 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

8.402 HxSegmentation2d Class Reference

A segmentation of a 2D image.

`#include <HxSegmentation2d.h>`

Public Methods

- **HxSegmentation2d** ()
  Constructor.
- **HxSegmentation2d** (**HxImageRep** inputIm, **HxImageRep** labIm)
  Constructor.
- ~**HxSegmentation2d** ()
  Destructor.
- int **ident** () const
  The identifier of this segmentation.
- void **setInputImage** (**HxImageRep** inputIm)
  Set the orginal image.
- **HxImageRep getInputImage** () const
  Get the orginal image.
- void **setLabeledImage** (**HxImageRep** labIm)
  Set the labeled image.
- **HxImageRep getLabeledImage** () const
  Get the labeled image.
- void **addBlob** (**HxBlob2d** *blob)
  Add a blob to the segmentation.
- void **addBlob** (int label, int xmin, int ymin, int width, int height)
  Add a blob to the segmentation.
- **HxBlob2dListConstIter getBlobBegin** () const

Get begin of the blob list.

- **HxBlob2dListConstIter getBlobEnd** () const

  Get end (STL term) of the blob list.

- **HxBlob2dListBackInserter getBlobInserter** ()

  Get a backinserter for the blob list.

- void **addRelation** (**HxString** name, **HxBlob2dRelation** ∗rel)

  Add a relation to the segmentation.

- **HxBlob2dRelation** ∗ **getRelation** (**HxString** name) const

  Get a relation from the segmentation.

- STD_OSTREAM & **put** (STD_OSTREAM &os) const

  Put segmentation on stream.

8.402.1 Detailed Description

A segmentation of a 2D image.

Basically, a list of **HxBlob2d** (p. 402)'s, possibly with relationships.

8.402.2 Constructor & Destructor Documentation

8.402.2.1 HxSegmentation2d::HxSegmentation2d ()

Constructor.

```
17 {
18     _ident = _nr++;
19 }
```

8.402.2.2 HxSegmentation2d::HxSegmentation2d (HxImageRep *inputIm*, HxImageRep *labIm*)

Constructor.

```
22 {
23     _inputImage = inputIm;
24     _labImage = labIm;
25     _ident = _nr++;
26 }
```

8.402.2.3 HxSegmentation2d::∼HxSegmentation2d ()

Destructor.

```
29 {
30     for(int i=0 ; i<_blobs.size() ; i++)
31         delete _blobs[i];
32     _blobs.clear();
33 }
```

8.402.3 Member Function Documentation

8.402.3.1 int HxSegmentation2d::ident () const

The identifier of this segmentation.

```
37 {
38     return _ident;
39 }
```

8.402.3.2 void HxSegmentation2d::setInputImage (HxImageRep *inputIm*)

Set the orginal image.

```
43 {
44     _inputImage = inputIm;
45 }
```

8.402.3.3 HxImageRep HxSegmentation2d::getInputImage () const

Get the orginal image.

```
49 {
50     return _inputImage;
51 }
```

8.402.3.4 void HxSegmentation2d::setLabeledImage (HxImageRep *labIm*)

Set the labeled image.

```
55 {
56     _labImage = labIm;
57 }
```

8.402.3.5 HxImageRep HxSegmentation2d::getLabeledImage () const

Get the labeled image.

```
61 {
62     return _labImage;
63 }
```

8.402.3.6 void HxSegmentation2d::addBlob (HxBlob2d * *blob*)

Add a blob to the segmentation.

Will take ownership of the blob.

```
67 {
68     _blobs.push_back(blob);
69 }
```

8.402.3.7 void HxSegmentation2d::addBlob (int *label*, int *xmin*, int *ymin*, int *width*, int *height*)

Add a blob to the segmentation.

```
73 {
74     HxBlob2d* blob = new HxBlob2d(label, xmin, ymin, width, height);
75     _blobs.push_back(blob);
76 }
```

8.402.3.8 HxBlob2dListConstIter HxSegmentation2d::getBlobBegin () const

Get begin of the blob list.

```
80 {
81     return _blobs.begin();
82 }
```

8.402.3.9 HxBlob2dListConstIter HxSegmentation2d::getBlobEnd () const

Get end (STL term) of the blob list.

```
86 {
87     return _blobs.end();
88 }
```

8.402.3.10 HxBlob2dListBackInserter HxSegmentation2d::getBlobInserter ()

Get a backinserter for the blob list.

```
92 {
93     return std::back_inserter(_blobs);
94 }
```

8.402.3.11 void HxSegmentation2d::addRelation (HxString *name*, HxBlob2dRelation * *rel*)

Add a relation to the segmentation.

Will take ownership of the relation.

```
98 {
99     _relations[name] = rel;
100 }
```

8.402.3.12 HxBlob2dRelation * HxSegmentation2d::getRelation (HxString *name*) const

Get a relation from the segmentation.

```
104 {
105     return (*_relations.find(name)).second;
106 }
```

8.402.3.13 STD_OSTREAM & HxSegmentation2d::put (STD_OSTREAM & *os*) const

Put segmentation on stream.

```
110 {
111     os << "Segmentation " << _ident << ", nr of blobs : " << _blobs.size()
112        << STD_ENDL;
113     for (int i=0 ; i<_blobs.size() ; i++)
114         os << (*_blobs[i]);
115     return os;
116 }
```

The documentation for this class was generated from the following files:

- **HxSegmentation2d.h**
- HxSegmentation2d.c

8.403 HxSF Class Reference

Class definition of the structuring function used for Mathematical Morphology operations.

`#include <HxSF.h>`

Public Methods

- **HxImageRep getKernel** () const
- **HxImageRep getHorizontalKernel** () const
- **HxImageRep getVerticalKernel** () const
- bool **isSeparable** () const
- bool **isSymetric** () const
- int **getConnectivity** () const
- HxSF **dilateSF** (int n)

  perform a "n-1" dilation or Minkowski addition of the SF.

- HxSF **erodeSF** (int n)
- HxSF **rotateSF** (int theta, bool sense)

  perform rotation of the SF; this is used in Thickening and Thinning the allowed values for theta are 45, 90, 180 sense is TRUE for COUNTERCLOCKWISE and FALSE for CLOCKWISE.

Constructors

- **HxSF** ()

Null Structuring Function.

- **HxSF** (const HxSF &)
  Copy Constructor.

- **HxSF** (**HxImageRep** im, bool sym, bool decomp, int conn)
- **HxSF** (**HxImageRep** imH, **HxImageRep** imV, bool sym, bool decomp, int conn)
- **HxSF** (**HxImageRep** im, **HxImageRep** imH, **HxImageRep** imV, bool sym, bool decomp, int conn)

Destructor

- ~**HxSF** ()

Operators

- HxSF & **operator=** (const HxSF &)
  Assignment operator.

Friends

- class **HxSFFactory**

8.403.1 Detailed Description

Class definition of the structuring function used for Mathematical Morphology operations.

Author:
Leon Todoran (`todoran@science.uva.nl`)

Version:
0.1

Date:
10.01.2002

the structuring element should have the same signature as the argument image

Remarks:
is it allowed for the se to have:

- even size?
- different x and y sizes?
- different signature than the image?

Todo:
still have to implement: -roatation of se

8.403.2 Constructor & Destructor Documentation

8.403.2.1 HxSF::HxSF ()

Null Structuring Function.

```
18 {
19     symetric        = false;
20     separable       = false;
21     connectivity    = 8;
22 }
```

8.403.2.2 HxSF::HxSF (const HxSF & *a*)

Copy Constructor.

```
26 {
27     se              = a.getKernel();
28     seH             = a.getHorizontalKernel();
29     seV             = a.getVerticalKernel();
30     symetric        = a.isSymetric();
31     separable       = a.isSeparable();
32     connectivity    = a.getConnectivity();
33 }
```

8.403.3 Member Function Documentation

8.403.3.1 HxSF & HxSF::operator= (const HxSF & *a*)

Assignment operator.

```
73 {
74     se              = a.getKernel();
75     seH             = a.getHorizontalKernel();
76     seV             = a.getVerticalKernel();
77     symetric        = a.isSymetric();
78     separable       = a.isSeparable();
79     connectivity    = a.getConnectivity();
80
81     return *this;
82 }
```

8.403.3.2 HxSF HxSF::dilateSF (int *n*)

perform a ”n-1” dilation or Minkowski addition of the SF.

```
111 {
112     if(n==0)
113         return *this;
114
115     size = se.sizes();
116     int increaseFactorX = size.x()/2;
117     int increaseFactorY = size.y()/2;
118     int newx = size.x()+2*n*increaseFactorX;
```

```
119     int newy = size.y()+2*n*increaseFactorY;
120     HxSizes sz(newx,newy,1);
121     HxPoint p(newx/2-size.x()/2,newy/2-(size.y())/2,0);
122
123     HxImageRep im = HxExtendVal(se,sz,0,p);
124
125     for(int i=0; i< n; i++)
126         im = HxDilation(im, *this);
127
128
129     HxSF sf(im, this->isSymetric(), false, this->getConnectivity() );
130
131     return sf;
132 }
```

8.403.3.3 HxSF HxSF::rotateSF (int *theta*, bool *sense*)

perform rotation of the SF; this is used in Thickening and Thinning the allowed values for theta are 45, 90, 180 sense is TRUE for COUNTERCLOCKWISE and FALSE for CLOCKWISE.

The documentation for this class was generated from the following files:

- **HxSF.h**
- HxSF.c

8.404 HxSFFactory Class Reference

Copyright (c) 2002, University of Amsterdam, The Netherlands.

```
#include <HxSFFactory.h>
```

Public Methods

- **HxSF makeSFfromImage** (**HxImageRep** im)

  New Structuring Function from a given image.

- **HxSF fromFunction** (**HxImageRep** im)

  New Structuring Function from a given function.

- **HxSF makeFlatSF** (**HxImageSignature** sig, **HxSizes** sz, **HxValue** val=0)

  New Flat Structuring Function of a given signature, size, and pixel value.

- **HxSF makeBoxSF** (**HxImageSignature** sig, **HxSizes** sz, **HxValue** val=0)
- **HxSF makeCrossSF** (**HxImageSignature** sig, **HxSizes** sz, **HxValue** val=0)
- **HxSF makeDiskSF** (**HxImageSignature** sig, **HxSizes** sz, **HxValue** val=0)
- **HxSF makeDiamondSF** (**HxImageSignature** sig, **HxSizes** sz, **HxValue** val=0)
- **HxSF makeGaussianSF** (**HxSizes** sz, double sigma)
- **HxSF makeParabolaSF** (**HxSizes** sz, double sigma)

Static Public Methods

- HxSFFactory & **instance** ()

  The one and only instance of this class.

8.404.1 Detailed Description

Copyright (c) 2002, University of Amsterdam, The Netherlands.

All rights reserved.

Author:
Dennis Koelma (koelma@science.uva.nl)
Leon Todoran (todoran@science.uva.nl)

Version:
0.1

8.404.2 Member Function Documentation

8.404.2.1 HxSFFactory & HxSFFactory::instance () [static]

The one and only instance of this class.

```
9 {
10     static HxSFFactory theFactory;
11     return theFactory;
12 }
```

8.404.2.2 HxSF HxSFFactory::makeSFfromImage (HxImageRep *kernel*)

New Structuring Function from a given image.

```
16 {
17      return HxSF(kernel, false, false, 8);
18 }
```

8.404.2.3 HxSF HxSFFactory::fromFunction (HxImageRep *im*)

New Structuring Function from a given function.

For instance, one can specify a Gaussian function here From the given function, the actuall image repesentation will be computed

8.404.2.4 HxSF HxSFFactory::makeFlatSF (HxImageSignature *sig*, HxSizes *sz*, HxValue *val* = 0)

New Flat Structuring Function of a given signature, size, and pixel value.

```
22 {
23     HxImageRep kernel = HxImageFactory::instance().fromValue(sig, sz, val);
24
25     //this is separable, so create the H and V kernels
26     HxSizes szV(sz.y(), 1, sz.z());
27     HxSizes szH(sz.x(), 1, sz.z());
28     HxImageRep hKernel = HxMakeFromValue(sig, szH, val);
29     HxImageRep vKernel = HxMakeFromValue(sig, szV, val);
30
31
32      return HxSF(kernel, hKernel, vKernel, true, true, 8);
33 }
```

The documentation for this class was generated from the following files:

- **HxSFFactory.h**
- HxSFFactory.c

8.405 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.

8.405.1 Detailed Description

Class definition for list of HxString's.

Specialization of list from STL.

8.405.2 Member Typedef Documentation

8.405.2.1 typedef std::back_insert_iterator<HxStringList> HxStringList::back_insert_iterator

back inserter.

8.405.3 Member Function Documentation

8.405.3.1 HxStringList & HxStringList::operator<< (const HxString & *s*) [inline]

Add string to the list.

```
52 {
53     push_back(s);
54     return *this;
55 }
```

8.405.3.2 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:

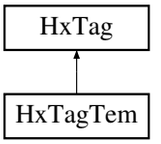
- **HxStringList.h**

8.406 HxTag Class Reference

Base class for tags.

`#include <HxTag.h>`

Inheritance diagram for HxTag::



Public Methods

- **HxTag** (**HxString** name)

  Constructor.

- virtual ~**HxTag** ()

Destructor.

- virtual HxTag * **clone** () const=0
  Clone operation.
- **HxString getName** () const
  Get the name of this tag.
- virtual std::ostream & **put** (std::ostream &) const=0
  To put the tag on an ostream.

Protected Methods

- **HxTag** (const HxTag &)
  Copy constructor.

8.406.1 Detailed Description

Base class for tags.

8.406.2 Constructor & Destructor Documentation

8.406.2.1 HxTag::HxTag (HxString *name*) [inline]

Constructor.

```
65     : _name(name)
66 {
67 }
```

8.406.2.2 HxTag::~HxTag () [virtual]

Destructor.

```
18 {
19 #ifdef CD_TRACE
20     HxEnvironment::instance()->outputStream()
21         << "HxTag::~HxTag()" << STD_ENDL;
22     HxEnvironment::instance()->flush();
23 #endif
24 }
```

8.406.2.3 HxTag::HxTag (const HxTag & *rhs*) [inline, protected]

Copy constructor.

```
59     : _name(rhs._name)
60 {
61 }
```

8.406.3 Member Function Documentation

8.406.3.1 virtual HxTag* HxTag::clone () const [pure virtual]

Clone operation.

Reimplemented in **HxTagTem** (p. 1203).

8.406.3.2 HxString HxTag::getName () const [inline]

Get the name of this tag.

```
71 {
72     return _name;
73 }
```

8.406.3.3 virtual std::ostream& HxTag::put (std::ostream & *os*) const [pure virtual]

To put the tag on an ostream.

Reimplemented in **HxTagTem** (p. 1204).

The documentation for this class was generated from the following files:

- **HxTag.h**
- HxTag.c

8.407 HxTag1Phase Struct Reference

1 phase pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

8.407.1 Detailed Description

1 phase pixel operation.

8.407.2 Member Function Documentation

8.407.2.1 HxString HxTag1Phase::toString () [inline]

Convert tag to string.

```
25 { return "1 phase"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.408 HxTag2Phase Struct Reference

2 phase pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

8.408.1 Detailed Description

2 phase pixel operation.

8.408.2 Member Function Documentation

8.408.2.1 HxString HxTag2Phase::toString () `[inline]`

Convert tag to string.

```
31 { return "2 phase"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.409 HxTagCnum Struct Reference

Coordinate enumerated neighbourhood iterator.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

8.409.1 Detailed Description

Coordinate enumerated neighbourhood iterator.

8.409.2 Member Function Documentation

8.409.2.1 HxString HxTagCnum::toString () [inline]

Convert tag to string.

```
76 { return "Coordinate enumerated"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.410 HxTagList Class Reference

A list of tags.

`#include <HxTagList.h>`

Public Types

- typedef **HxTag** * **TagPtr**
  Type definition for a pointer to an **HxTag** (p. 1192).
- typedef std::list< **TagPtr** > **List**
  Type definition for a list of tag pointers.

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. 366).
- **HxTag** * **getTag** (**HxString** name) const
  Get a tag, use via **HxGetTag** (p. 366).

- **HxString toString** () const

  Make a sttring with names of all tags.

- std::ostream & **put** (std::ostream &) const

  To put the list on an ostream.

8.410.1 Detailed Description

A list of tags.

Basically, a tag is a combination of a name and a value. The name is represented by an HxString and the value is represented by an instantiation of **HxTagTem** (p. 1202), which is a specialization of **HxTag** (p. 1192). A value can be anything as long as it can be copy-constructed.

Use **HxAddTag** (p. 366) to add tags to the list. Use **HxGetTag**(const HxTagList&,HxString) (p. 366) and **HxGetTag**(const HxTagList&,HxString,ValT) (p. 366) to obtain values stored in the list. Use **HxTagIsSet** (p. 366) to check whether a tag is in the list.

8.410.2 Member Typedef Documentation

8.410.2.1 typedef HxTag* HxTagList::TagPtr

Type definition for a pointer to an **HxTag** (p. 1192).

8.410.2.2 typedef std::list<TagPtr> HxTagList::List

Type definition for a list of tag pointers.

8.410.3 Constructor & Destructor Documentation

8.410.3.1 HxTagList::HxTagList () [inline]

Constructor.

```
38 {}
```

8.410.3.2 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 }
```

8.410.3.3 HxTagList::~HxTagList ()

Destructor.

```
39 {
40     List::iterator ptr;
41
42     for (ptr = _list.begin(); ptr != _list.end(); ptr++)
43         delete (*ptr);
44 }
```

8.410.4 Member Function Documentation

8.410.4.1 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 }
```

8.410.4.2 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 }
```

8.410.4.3 void HxTagList::addTag (HxTag * *tag*)

Add a tag, use via **HxAddTag** (p. 366).

```
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 }
```

8.410.4.4 HxTag * HxTagList::getTag (HxString *name*) const

Get a tag, use via **HxGetTag** (p. 366).

```
90 {
91     List::const_iterator ptr = findTag(name);
92     return ptr != _list.end() ? (*ptr) : 0;
93 }
```

8.410.4.5 HxString HxTagList::toString () const

Make a sttring with names of all tags.

```
97 {
98     int i = 0;
99     HxString s("(");
100     List::const_iterator ptr;
101     for (ptr = _list.begin(); ptr != _list.end(); ptr++)
102     {
103         if (i > 0)
104             s += ", ";
105         s += (*ptr)->getName();
106 
107     }
108     s += ")";
109     return s;
110 }
```

8.410.4.6 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

8.411 HxTagLoop Struct Reference

Loop neighbourhood iterator.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

8.411.1 Detailed Description

Loop neighbourhood iterator.

8.411.2 Member Function Documentation

8.411.2.1 HxString HxTagLoop::toString () [inline]

Convert tag to string.

```
82 { return "Loop"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.412 HxTagNPhase Struct Reference

N phase pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

8.412.1 Detailed Description

N phase pixel operation.

8.412.2 Member Function Documentation

8.412.2.1 HxString HxTagNPhase::toString () [inline]

Convert tag to string.

```
37 { return "N phase"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.413 HxTagPixOpIn Struct Reference

In pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

8.413.1 Detailed Description

In pixel operation.

8.413.2 Member Function Documentation

8.413.2.1 HxString HxTagPixOpIn::toString () [inline]

Convert tag to string.

```
61 { return "In"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.414 HxTagPixOpOut Struct Reference

Out pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

8.414.1 Detailed Description

Out pixel operation.

8.414.2 Member Function Documentation

8.414.2.1 HxString HxTagPixOpOut::toString () [inline]

Convert tag to string.

```
67 { return "Out"; }
```

The documentation for this struct was generated from the following file:

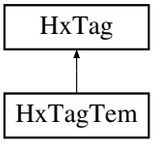
- **HxCategories.h**

8.415 HxTagTem Class Template Reference

Template specialization of **HxTag** (p. 1192) to store any type of value as a tag.

`#include <HxTagTem.h>`

Inheritance diagram for HxTagTem::



Public Methods

- **HxTagTem** (**HxString** name, ValT v)
  Constructor.
- virtual ~**HxTagTem** ()
  Destructor.
- virtual **HxTag** * **clone** () const
  Clone operation.
- ValT **getValue** () const
  Get the value of this tag.
- virtual std::ostream & **put** (std::ostream &) const
  To put the tag on an ostream.

Protected Methods

- **HxTagTem** (const HxTagTem &rhs)
  Copy constructor.

8.415.1 Detailed Description

template<class ValT> class HxTagTem< ValT >

Template specialization of **HxTag** (p. 1192) to store any type of value as a tag.

8.415.2 Constructor & Destructor Documentation

8.415.2.1 template<class ValT> HxTagTem< ValT >::HxTagTem (HxString *name*, ValT *v*) [inline]

Constructor.

```
51     : HxTag(name), _value(v)
52 {
53 }
```

8.415.2.2 template<class ValT> HxTagTem< ValT >::~HxTagTem () [virtual]

Destructor.

```
20 {
21 }
```

8.415.2.3 template<class ValT> HxTagTem< ValT >::HxTagTem (const HxTagTem< ValT > & *rhs*) [inline, protected]

Copy constructor.

```
58     :  HxTag(rhs), _value(rhs._value)
59 {
60 }
```

8.415.3 Member Function Documentation

8.415.3.1 template<class ValT> HxTag * HxTagTem< ValT >::clone () const [virtual]

Clone operation.

Reimplemented from **HxTag** (p. 1194).

```
26 {
27     return new HxTagTem(*this);
28 }
```

8.415.3.2 template<class ValT> ValT HxTagTem< ValT >::getValue () const [inline]

Get the value of this tag.

```
35 {return _value;}
```

8.415.3.3 template<class ValT> std::ostream & HxTagTem< ValT >::put (std::ostream & *os*) const [virtual]

To put the tag on an ostream.

Reimplemented from **HxTag** (p. 1194).

```
33 {
34     os << getName() << " = " << _value;
35     return os;
36 }
```

The documentation for this class was generated from the following files:

- **HxTagTem.h**
- HxTagTem.c

8.416 HxTagTransInVar Struct Reference

Translation invariant pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()

  Convert tag to string.

8.416.1 Detailed Description

Translation invariant pixel operation.

8.416.2 Member Function Documentation

8.416.2.1 HxString HxTagTransInVar::toString () [inline]

Convert tag to string.

```
52 { return "Translation invariant"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.417 HxTagTransVar Struct Reference

Translation variant pixel operation.

`#include <HxCategories.h>`

Public Methods

- **HxString toString** ()
  Convert tag to string.

8.417.1 Detailed Description

Translation variant pixel operation.

8.417.2 Member Function Documentation

8.417.2.1 HxString HxTagTransVar::toString () [inline]

Convert tag to string.

```
46 { return "Translation variant"; }
```

The documentation for this struct was generated from the following file:

- **HxCategories.h**

8.418 HxUpoAbs Class Template Reference

Pixel functor for computation of absolute value.

`#include <HxUpoAbs.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoAbs** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.abs() #.

Static Public Methods

- **HxString className** ()
  The name : "abs".

8.418.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAbs< DstValT, SrcValT >

Pixel functor for computation of absolute value.

8.418.2 Member Typedef Documentation

8.418.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoAbs::TransVarianceCategory

Functor is translation invariant.

8.418.3 Constructor & Destructor Documentation

8.418.3.1 template<class DstValT, class SrcValT> HxUpoAbs< DstValT, SrcValT >::HxUpoAbs (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.418.4 Member Function Documentation

8.418.4.1 template<class DstValT, class SrcValT> DstValT HxUpoAbs< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.abs() #.

```
33                          { return x.abs(); }
```

8.418.4.2 template<class DstValT, class SrcValT> HxString HxUpoAbs< DstValT, SrcValT >::className () [inline, static]

The name : ”abs”.

```
37                          { return HxString("abs"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAbs.h**

8.419 HxUpoAcos Class Template Reference

Pixel functor for computation of arc cosine.

```
#include <HxUpoAcos.h>
```

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoAcos** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.acos() #.

Static Public Methods

- **HxString className** ()
  The name : "acos".

8.419.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAcos< DstValT, SrcValT >

Pixel functor for computation of arc cosine.

8.419.2 Member Typedef Documentation

8.419.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoAcos::TransVarianceCategory

Functor is translation invariant.

8.419.3 Constructor & Destructor Documentation

8.419.3.1 template<class DstValT, class SrcValT> HxUpoAcos< DstValT, SrcValT >::HxUpoAcos (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.419.4 Member Function Documentation

8.419.4.1 template<class DstValT, class SrcValT> DstValT HxUpoAcos< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.acos() #.

```
33                          { return x.acos(); }
```

8.419.4.2 template<class DstValT, class SrcValT> HxString HxUpoAcos< DstValT, SrcValT >::className () [inline, static]

The name : ”acos”.

```
37                          { return HxString("acos"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAcos.h**

8.420 HxUpoArg Class Template Reference

Pixel functor for computation of argument.

`#include <HxUpoArg.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoArg** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.arg() #.

Static Public Methods

- **HxString className** ()
  The name : ”arg”.

8.420.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoArg< DstValT, SrcValT >

Pixel functor for computation of argument.

8.420.2 Member Typedef Documentation

8.420.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoArg::TransVarianceCategory

Functor is translation invariant.

8.420.3 Constructor & Destructor Documentation

8.420.3.1 template<class DstValT, class SrcValT> HxUpoArg< DstValT, SrcValT >::HxUpoArg (HxTagList &) [inline]

Constructor : empty.

```
28                          {}
```

8.420.4 Member Function Documentation

8.420.4.1 template<class DstValT, class SrcValT> DstValT HxUpoArg< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.arg() #.

```
32                          { return x.arg(); }
```

8.420.4.2 template<class DstValT, class SrcValT> HxString HxUpoArg< DstValT, SrcValT >::className () [inline, static]

The name : ”arg”.

```
36                          { return HxString("arg"); }
```

The documentation for this class was generated from the following file:

- **HxUpoArg.h**

8.421 HxUpoAsin Class Template Reference

Pixel functor for computation of arc sine.

`#include <HxUpoAsin.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoAsin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.asin() #.

Static Public Methods

- **HxString className** ()
  The name : "asin".

8.421.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAsin< DstValT, SrcValT >

Pixel functor for computation of arc sine.

8.421.2 Member Typedef Documentation

8.421.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoAsin::TransVarianceCategory

Functor is translation invariant.

8.421.3 Constructor & Destructor Documentation

8.421.3.1 template<class DstValT, class SrcValT> HxUpoAsin< DstValT, SrcValT >::HxUpoAsin (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.421.4 Member Function Documentation

8.421.4.1 template<class DstValT, class SrcValT> DstValT HxUpoAsin< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.asin() #.

```
33                          { return x.asin(); }
```

8.421.4.2 template<class DstValT, class SrcValT> HxString HxUpoAsin< DstValT, SrcValT >::className () [inline, static]

The name : ”asin”.

```
37                          { return HxString("asin"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAsin.h**

8.422 HxUpoAtan Class Template Reference

Pixel functor for computation of arc tangent.

`#include <HxUpoAtan.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoAtan** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.atan() #.

Static Public Methods

- **HxString className** ()
  The name : ”atan”.

8.422.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAtan< DstValT, SrcValT >

Pixel functor for computation of arc tangent.

8.422.2 Member Typedef Documentation

8.422.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoAtan::TransVarianceCategory

Functor is translation invariant.

8.422.3 Constructor & Destructor Documentation

8.422.3.1 template<class DstValT, class SrcValT> HxUpoAtan< DstValT, SrcValT >::HxUpoAtan (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.422.4 Member Function Documentation

8.422.4.1 template<class DstValT, class SrcValT> DstValT HxUpoAtan< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.atan() #.

```
33                          { return x.atan(); }
```

8.422.4.2 template<class DstValT, class SrcValT> HxString HxUpoAtan< DstValT, SrcValT >::className () [inline, static]

The name : ”atan”.

```
37                          { return HxString("atan"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAtan.h**

8.423 HxUpoAtan2 Class Template Reference

Pixel functor for computation of arc tangent.

`#include <HxUpoAtan2.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoAtan2** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.atan2() #.

Static Public Methods

- **HxString className** ()
  The name : "atan2".

8.423.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoAtan2< DstValT, SrcValT >

Pixel functor for computation of arc tangent.

8.423.2 Member Typedef Documentation

8.423.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoAtan2::TransVarianceCategory

Functor is translation invariant.

8.423.3 Constructor & Destructor Documentation

8.423.3.1 template<class DstValT, class SrcValT> HxUpoAtan2< DstValT, SrcValT >::HxUpoAtan2 (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.423.4 Member Function Documentation

8.423.4.1 template<class DstValT, class SrcValT> DstValT HxUpoAtan2< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.atan2() #.

```
33                          { return x.atan2(); }
```

8.423.4.2 template<class DstValT, class SrcValT> HxString HxUpoAtan2< DstValT, SrcValT >::className () [inline, static]

The name : ”atan2”.

```
37                          { return HxString("atan2"); }
```

The documentation for this class was generated from the following file:

- **HxUpoAtan2.h**

8.424 HxUpoCeil Class Template Reference

Pixel functor for computation of ceiling.

`#include <HxUpoCeil.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoCeil** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.ceil() #.

Static Public Methods

- **HxString className** ()
  The name : ”ceil”.

8.424.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoCeil< DstValT, SrcValT >

Pixel functor for computation of ceiling.

8.424.2 Member Typedef Documentation

8.424.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoCeil::TransVarianceCategory

Functor is translation invariant.

8.424.3 Constructor & Destructor Documentation

8.424.3.1 template<class DstValT, class SrcValT> HxUpoCeil< DstValT, SrcValT >::HxUpoCeil (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.424.4 Member Function Documentation

8.424.4.1 template<class DstValT, class SrcValT> DstValT HxUpoCeil< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.ceil() #.

```
33                          { return x.ceil(); }
```

8.424.4.2 template<class DstValT, class SrcValT> HxString HxUpoCeil< DstValT, SrcValT >::className () [inline, static]

The name : ”ceil”.

```
37                          { return HxString("ceil"); }
```

The documentation for this class was generated from the following file:

- **HxUpoCeil.h**

8.425 HxUpoColSpace Class Template Reference

Pixel functor for color space conversion.

`#include <HxUpoColSpace.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoColSpace** (**HxTagList** &tags)
  Constructor.
- DstValT **doIt** (const SrcValT &x)
  Actual operation :.

Static Public Methods

- **HxString className** ()
  The name : "colorSpace".

8.425.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoColSpace< DstValT, SrcValT >

Pixel functor for color space conversion.

8.425.2 Member Typedef Documentation

8.425.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoColSpace::TransVarianceCategory

Functor is translation invariant.

8.425.3 Constructor & Destructor Documentation

8.425.3.1 template<class DstValT, class SrcValT> HxUpoColSpace< DstValT, SrcValT >::HxUpoColSpace (HxTagList & *tags*) [inline]

Constructor.

```
29                          {
30                              _fromSpace = HxGetTag<HxColorModel>(tags,
31                                  "fromColorSpace");
32                              _toSpace = HxGetTag<HxColorModel>(tags,
33                                  "toColorSpace");
34                          }
```

8.425.4 Member Function Documentation

8.425.4.1 template<class DstValT, class SrcValT> DstValT HxUpoColSpace< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation :.

```
38                          { return HxColor(x,_fromSpace).convert(_toSpace).value();}
```

8.425.4.2 template<class DstValT, class SrcValT> HxString HxUpoColSpace< DstValT, SrcValT >::className () [inline, static]

The name : ”colorSpace”.

```
42                          { return HxString("colorSpace"); }
```

The documentation for this class was generated from the following file:

- **HxUpoColSpace.h**

8.426 HxUpoComplement Class Template Reference

Pixel functor for computation of comlement.

`#include <HxUpoComplement.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoComplement** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.complement() #.

Static Public Methods

- **HxString className** ()
  The name : ”complement”.

8.426.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoComplement< DstValT, SrcValT >

Pixel functor for computation of comlement.

8.426.2 Member Typedef Documentation

8.426.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoComplement::TransVarianceCategory

Functor is translation invariant.

8.426.3 Constructor & Destructor Documentation

8.426.3.1 template<class DstValT, class SrcValT> HxUpoComplement< DstValT, SrcValT >::HxUpoComplement (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.426.4 Member Function Documentation

8.426.4.1 template<class DstValT, class SrcValT> DstValT HxUpoComplement< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.complement() #.

```
33                          { return x.complement(); }
```

8.426.4.2 template<class DstValT, class SrcValT> HxString HxUpoComplement< DstValT, SrcValT >::className () [inline, static]

The name : ”complement”.

```
37                          { return HxString("complement"); }
```

The documentation for this class was generated from the following file:

- **HxUpoComplement.h**

8.427 HxUpoConjugate Class Template Reference

Pixel functor for computation of conjugate.

`#include <HxUpoConjugate.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoConjugate** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.conjugate() #.

Static Public Methods

- **HxString className** ()
  The name : "conjugate".

8.427.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoConjugate< DstValT, SrcValT >

Pixel functor for computation of conjugate.

8.427.2 Member Typedef Documentation

8.427.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoConjugate::TransVarianceCategory

Functor is translation invariant.

8.427.3 Constructor & Destructor Documentation

8.427.3.1 template<class DstValT, class SrcValT> HxUpoConjugate< DstValT, SrcValT >::HxUpoConjugate (HxTagList &) [inline]

Constructor : empty.

```
28                          {}
```

8.427.4 Member Function Documentation

8.427.4.1 template<class DstValT, class SrcValT> DstValT HxUpoConjugate< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.conjugate() #.

```
32                          { return x.conjugate(); }
```

8.427.4.2 template<class DstValT, class SrcValT> HxString HxUpoConjugate< DstValT, SrcValT >::className () [inline, static]

The name : ”conjugate”.

```
36                          { return HxString("conjugate"); }
```

The documentation for this class was generated from the following file:

- **HxUpoConjugate.h**

8.428 HxUpoCos Class Template Reference

Pixel functor for computation of cosine.

`#include <HxUpoCos.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoCos** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.cos() #.

Static Public Methods

- **HxString className** ()
  The name : ”cos”.

8.428.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoCos< DstValT, SrcValT >

Pixel functor for computation of cosine.

8.428.2 Member Typedef Documentation

8.428.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoCos::TransVarianceCategory

Functor is translation invariant.

8.428.3 Constructor & Destructor Documentation

8.428.3.1 template<class DstValT, class SrcValT> HxUpoCos< DstValT, SrcValT >::HxUpoCos (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.428.4 Member Function Documentation

8.428.4.1 template<class DstValT, class SrcValT> DstValT HxUpoCos< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.cos() #.

```
33                          { return x.cos(); }
```

8.428.4.2 template<class DstValT, class SrcValT> HxString HxUpoCos< DstValT, SrcValT >::className () [inline, static]

The name : ”cos”.

```
37                          { return HxString("cos"); }
```

The documentation for this class was generated from the following file:

- **HxUpoCos.h**

8.429 HxUpoCosh Class Template Reference

Pixel functor for computation of hyperbolic consine.

`#include <HxUpoCosh.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoCosh** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.cosh() #.

Static Public Methods

- **HxString className** ()
  The name : "cosh".

8.429.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoCosh< DstValT, SrcValT >

Pixel functor for computation of hyperbolic consine.

8.429.2 Member Typedef Documentation

8.429.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoCosh::TransVarianceCategory

Functor is translation invariant.

8.429.3 Constructor & Destructor Documentation

8.429.3.1 template<class DstValT, class SrcValT> HxUpoCosh< DstValT, SrcValT >::HxUpoCosh (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.429.4 Member Function Documentation

8.429.4.1 template<class DstValT, class SrcValT> DstValT HxUpoCosh< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.cosh() #.

```
33                          { return x.cosh(); }
```

8.429.4.2 template<class DstValT, class SrcValT> HxString HxUpoCosh< DstValT, SrcValT >::className () [inline, static]

The name : ”cosh”.

```
37                          { return HxString("cosh"); }
```

The documentation for this class was generated from the following file:

- **HxUpoCosh.h**

8.430 HxUpoExp Class Template Reference

Pixel functor for computation of exponent.

`#include <HxUpoExp.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoExp** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.exp() #.

Static Public Methods

- **HxString className** ()
  The name : ”exp”.

8.430.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoExp< DstValT, SrcValT >

Pixel functor for computation of exponent.

8.430.2 Member Typedef Documentation

8.430.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoExp::TransVarianceCategory

Functor is translation invariant.

8.430.3 Constructor & Destructor Documentation

8.430.3.1 template<class DstValT, class SrcValT> HxUpoExp< DstValT, SrcValT >::HxUpoExp (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.430.4 Member Function Documentation

8.430.4.1 template<class DstValT, class SrcValT> DstValT HxUpoExp< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.exp() #.

```
33                          { return x.exp(); }
```

8.430.4.2 template<class DstValT, class SrcValT> HxString HxUpoExp< DstValT, SrcValT >::className () [inline, static]

The name : ”exp”.

```
37                          { return HxString("exp"); }
```

The documentation for this class was generated from the following file:

- **HxUpoExp.h**

8.431 HxUpoFloor Class Template Reference

Pixel functor for computation of floor.

`#include <HxUpoFloor.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoFloor** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.floor() #.

Static Public Methods

- **HxString className** ()
  The name : "floor".

8.431.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoFloor< DstValT, SrcValT >

Pixel functor for computation of floor.

8.431.2 Member Typedef Documentation

8.431.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoFloor::TransVarianceCategory

Functor is translation invariant.

8.431.3 Constructor & Destructor Documentation

8.431.3.1 template<class DstValT, class SrcValT> HxUpoFloor< DstValT, SrcValT >::HxUpoFloor (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.431.4 Member Function Documentation

8.431.4.1 template<class DstValT, class SrcValT> DstValT HxUpoFloor< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.floor() #.

```
33                          { return x.floor(); }
```

8.431.4.2 template<class DstValT, class SrcValT> HxString HxUpoFloor< DstValT, SrcValT >::className () [inline, static]

The name : ”floor”.

```
37                          { return HxString("floor"); }
```

The documentation for this class was generated from the following file:

- **HxUpoFloor.h**

8.432 HxUpoLog Class Template Reference

Pixel functor for computation of natural logarithm.

`#include <HxUpoLog.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoLog** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.log() #.

Static Public Methods

- **HxString className** ()
  The name : ”log”.

8.432.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoLog< DstValT, SrcValT >

Pixel functor for computation of natural logarithm.

8.432.2 Member Typedef Documentation

8.432.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoLog::TransVarianceCategory

Functor is translation invariant.

8.432.3 Constructor & Destructor Documentation

8.432.3.1 template<class DstValT, class SrcValT> HxUpoLog< DstValT, SrcValT >::HxUpoLog (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.432.4 Member Function Documentation

8.432.4.1 template<class DstValT, class SrcValT> DstValT HxUpoLog< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.log() #.

```
33                          { return x.log(); }
```

8.432.4.2 template<class DstValT, class SrcValT> HxString HxUpoLog< DstValT, SrcValT >::className () [inline, static]

The name : ”log”.

```
37                          { return HxString("log"); }
```

The documentation for this class was generated from the following file:

- **HxUpoLog.h**

8.433 HxUpoLog10 Class Template Reference

Pixel functor for computation of base 10 logarithm.

`#include <HxUpoLog10.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoLog10** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.log10() #.

Static Public Methods

- **HxString className** ()
  The name : "log10".

8.433.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoLog10< DstValT, SrcValT >

Pixel functor for computation of base 10 logarithm.

8.433.2 Member Typedef Documentation

8.433.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoLog10::TransVarianceCategory

Functor is translation invariant.

8.433.3 Constructor & Destructor Documentation

8.433.3.1 template<class DstValT, class SrcValT> HxUpoLog10< DstValT, SrcValT >::HxUpoLog10 (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.433.4 Member Function Documentation

8.433.4.1 template<class DstValT, class SrcValT> DstValT HxUpoLog10< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.log10() #.

```
33                          { return x.log10(); }
```

8.433.4.2 template<class DstValT, class SrcValT> HxString HxUpoLog10< DstValT, SrcValT >::className () [inline, static]

The name : ”log10”.

```
37                          { return HxString("log10"); }
```

The documentation for this class was generated from the following file:

- **HxUpoLog10.h**

8.434 HxUpoMax Class Template Reference

Pixel functor for computation of maximum.

`#include <HxUpoMax.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoMax** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.max() #.

Static Public Methods

- **HxString className** ()
  The name : ”max”.

8.434.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoMax< DstValT, SrcValT >

Pixel functor for computation of maximum.

8.434.2 Member Typedef Documentation

8.434.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoMax::TransVarianceCategory

Functor is translation invariant.

8.434.3 Constructor & Destructor Documentation

8.434.3.1 template<class DstValT, class SrcValT> HxUpoMax< DstValT, SrcValT >::HxUpoMax (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.434.4 Member Function Documentation

8.434.4.1 template<class DstValT, class SrcValT> DstValT HxUpoMax< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.max() #.

```
33                          { return x.max(); }
```

8.434.4.2 template<class DstValT, class SrcValT> HxString HxUpoMax< DstValT, SrcValT >::className () [inline, static]

The name : ”max”.

```
37                          { return HxString("max"); }
```

The documentation for this class was generated from the following file:

- **HxUpoMax.h**

8.435 HxUpoMin Class Template Reference

Pixel functor for computation of minimum.

`#include <HxUpoMin.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoMin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.min() #.

Static Public Methods

- **HxString className** ()
  The name : "min".

8.435.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoMin< DstValT, SrcValT >

Pixel functor for computation of minimum.

8.435.2 Member Typedef Documentation

8.435.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoMin::TransVarianceCategory

Functor is translation invariant.

8.435.3 Constructor & Destructor Documentation

8.435.3.1 template<class DstValT, class SrcValT> HxUpoMin< DstValT, SrcValT >::HxUpoMin (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.435.4 Member Function Documentation

8.435.4.1 template<class DstValT, class SrcValT> DstValT HxUpoMin< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.min() #.

```
33                          { return x.min(); }
```

8.435.4.2 template<class DstValT, class SrcValT> HxString HxUpoMin< DstValT, SrcValT >::className () [inline, static]

The name : ”min”.

```
37                          { return HxString("min"); }
```

The documentation for this class was generated from the following file:

- **HxUpoMin.h**

8.436 HxUpoNegate Class Template Reference

Pixel functor for computation of negation.

`#include <HxUpoNegate.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoNegate** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return -x #.

Static Public Methods

- **HxString className** ()
  The name : ”negate”.

8.436.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNegate< DstValT, SrcValT >

Pixel functor for computation of negation.

8.436.2 Member Typedef Documentation

8.436.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoNegate::TransVarianceCategory

Functor is translation invariant.

8.436.3 Constructor & Destructor Documentation

8.436.3.1 template<class DstValT, class SrcValT> HxUpoNegate< DstValT, SrcValT >::HxUpoNegate (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.436.4 Member Function Documentation

8.436.4.1 template<class DstValT, class SrcValT> DstValT HxUpoNegate< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return -x #.

```
33                          { return -x; }
```

8.436.4.2 template<class DstValT, class SrcValT> HxString HxUpoNegate< DstValT, SrcValT >::className () [inline, static]

The name : ”negate”.

```
37                          { return HxString("negate"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNegate.h**

8.437 HxUpoNorm1 Class Template Reference

Pixel functor for computation of L1 norm.

`#include <HxUpoNorm1.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoNorm1** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.norm1() #.

Static Public Methods

- **HxString className** ()
  The name : "norm1".

8.437.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNorm1< DstValT, SrcValT >

Pixel functor for computation of L1 norm.

8.437.2 Member Typedef Documentation

8.437.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoNorm1::TransVarianceCategory

Functor is translation invariant.

8.437.3 Constructor & Destructor Documentation

8.437.3.1 template<class DstValT, class SrcValT> HxUpoNorm1< DstValT, SrcValT >::HxUpoNorm1 (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.437.4 Member Function Documentation

8.437.4.1 template<class DstValT, class SrcValT> DstValT HxUpoNorm1< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.norm1() #.

```
33                          { return x.norm1(); }
```

8.437.4.2 template<class DstValT, class SrcValT> HxString HxUpoNorm1< DstValT, SrcValT >::className () [inline, static]

The name : ”norm1”.

```
37                          { return HxString("norm1"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNorm1.h**

8.438 HxUpoNorm2 Class Template Reference

Pixel functor for computation of L2 norm.

`#include <HxUpoNorm2.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoNorm2** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.norm2() #.

Static Public Methods

- **HxString className** ()
  The name : ”norm2”.

8.438.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNorm2< DstValT, SrcValT >

Pixel functor for computation of L2 norm.

8.438.2 Member Typedef Documentation

8.438.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoNorm2::TransVarianceCategory

Functor is translation invariant.

8.438.3 Constructor & Destructor Documentation

8.438.3.1 template<class DstValT, class SrcValT> HxUpoNorm2< DstValT, SrcValT >::HxUpoNorm2 (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.438.4 Member Function Documentation

8.438.4.1 template<class DstValT, class SrcValT> DstValT HxUpoNorm2< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.norm2() #.

```
33                          { return x.norm2(); }
```

8.438.4.2 template<class DstValT, class SrcValT> HxString HxUpoNorm2< DstValT, SrcValT >::className () [inline, static]

The name : ”norm2”.

```
37                          { return HxString("norm2"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNorm2.h**

8.439 HxUpoNorm2Sqr Class Template Reference

Pixel functor for computation of squared L2 norm.

`#include <HxUpoNorm2Sqr.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

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".

8.439.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNorm2Sqr< DstValT, SrcValT >

Pixel functor for computation of squared L2 norm.

8.439.2 Member Typedef Documentation

8.439.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoNorm2Sqr::TransVarianceCategory

Functor is translation invariant.

8.439.3 Constructor & Destructor Documentation

8.439.3.1 template<class DstValT, class SrcValT> HxUpoNorm2Sqr< DstValT, SrcValT >::HxUpoNorm2Sqr (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.439.4 Member Function Documentation

8.439.4.1 template<class DstValT, class SrcValT> DstValT HxUpoNorm2Sqr< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return (x∗x).sum() #.

```
33                          { return (x*x).sum(); }
```

8.439.4.2 template<class DstValT, class SrcValT> HxString HxUpoNorm2Sqr< DstValT, SrcValT >::className () [inline, static]

The name : ”norm2sqr”.

```
37                          { return HxString("norm2sqr"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNorm2Sqr.h**

8.440 HxUpoNormInf Class Template Reference

Pixel functor for computation of L inf norm.

`#include <HxUpoNormInf.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoNormInf** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.normInf() #.

Static Public Methods

- **HxString className** ()
  The name : ”normInf”.

8.440.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoNormInf< DstValT, SrcValT >

Pixel functor for computation of L inf norm.

8.440.2 Member Typedef Documentation

8.440.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoNormInf::TransVarianceCategory

Functor is translation invariant.

8.440.3 Constructor & Destructor Documentation

8.440.3.1 template<class DstValT, class SrcValT> HxUpoNormInf< DstValT, SrcValT >::HxUpoNormInf (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.440.4 Member Function Documentation

8.440.4.1 template<class DstValT, class SrcValT> DstValT HxUpoNormInf< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.normInf() #.

```
33                          { return x.normInf(); }
```

8.440.4.2 template<class DstValT, class SrcValT> HxString HxUpoNormInf< DstValT, SrcValT >::className () [inline, static]

The name : "normInf".

```
37                          { return HxString("normInf"); }
```

The documentation for this class was generated from the following file:

- **HxUpoNormInf.h**

8.441 HxUpoProduct Class Template Reference

Pixel functor for computation of unary product.

`#include <HxUpoProduct.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoProduct** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.product() #.

Static Public Methods

- **HxString className** ()
  The name : "product".

8.441.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoProduct< DstValT, SrcValT >

Pixel functor for computation of unary product.

8.441.2 Member Typedef Documentation

8.441.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoProduct::TransVarianceCategory

Functor is translation invariant.

8.441.3 Constructor & Destructor Documentation

8.441.3.1 template<class DstValT, class SrcValT> HxUpoProduct< DstValT, SrcValT >::HxUpoProduct (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.441.4 Member Function Documentation

8.441.4.1 template<class DstValT, class SrcValT> DstValT HxUpoProduct< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.product() #.

```
33                          { return x.product(); }
```

8.441.4.2 template<class DstValT, class SrcValT> HxString HxUpoProduct< DstValT, SrcValT >::className () [inline, static]

The name : ”product”.

```
37                          { return HxString("product"); }
```

The documentation for this class was generated from the following file:

- **HxUpoProduct.h**

8.442 HxUpoRound Class Template Reference

Pixel functor for computation of round.

`#include <HxUpoRound.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoRound** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.round() #.

Static Public Methods

- **HxString className** ()
  The name : ”round”.

8.442.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoRound< DstValT, SrcValT >

Pixel functor for computation of round.

8.442.2 Member Typedef Documentation

8.442.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoRound::TransVarianceCategory

Functor is translation invariant.

8.442.3 Constructor & Destructor Documentation

8.442.3.1 template<class DstValT, class SrcValT> HxUpoRound< DstValT, SrcValT >::HxUpoRound (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.442.4 Member Function Documentation

8.442.4.1 template<class DstValT, class SrcValT> DstValT HxUpoRound< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.round() #.

```
33                          { return x.round(); }
```

8.442.4.2 template<class DstValT, class SrcValT> HxString HxUpoRound< DstValT, SrcValT >::className () [inline, static]

The name : ”round”.

```
37                          { return HxString("round"); }
```

The documentation for this class was generated from the following file:

- **HxUpoRound.h**

8.443 HxUpoSin Class Template Reference

Pixel functor for computation of sine.

`#include <HxUpoSin.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoSin** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.sin() #.

Static Public Methods

- **HxString className** ()
  The name : "sin".

8.443.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSin< DstValT, SrcValT >

Pixel functor for computation of sine.

8.443.2 Member Typedef Documentation

8.443.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoSin::TransVarianceCategory

Functor is translation invariant.

8.443.3 Constructor & Destructor Documentation

8.443.3.1 template<class DstValT, class SrcValT> HxUpoSin< DstValT, SrcValT >::HxUpoSin (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.443.4 Member Function Documentation

8.443.4.1 template<class DstValT, class SrcValT> DstValT HxUpoSin< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sin() #.

```
33                          { return x.sin(); }
```

8.443.4.2 template<class DstValT, class SrcValT> HxString HxUpoSin< DstValT, SrcValT >::className () [inline, static]

The name : ”sin”.

```
37                          { return HxString("sin"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSin.h**

8.444 HxUpoSinh Class Template Reference

Pixel functor for computation of hyperbolic sine.

`#include <HxUpoSinh.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoSinh** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.sinh() #.

Static Public Methods

- **HxString className** ()
  The name : ”sinh”.

8.444.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSinh< DstValT, SrcValT >

Pixel functor for computation of hyperbolic sine.

8.444.2 Member Typedef Documentation

8.444.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoSinh::TransVarianceCategory

Functor is translation invariant.

8.444.3 Constructor & Destructor Documentation

8.444.3.1 template<class DstValT, class SrcValT> HxUpoSinh< DstValT, SrcValT >::HxUpoSinh (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.444.4 Member Function Documentation

8.444.4.1 template<class DstValT, class SrcValT> DstValT HxUpoSinh< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sinh() #.

```
33                          { return x.sinh(); }
```

8.444.4.2 template<class DstValT, class SrcValT> HxString HxUpoSinh< DstValT, SrcValT >::className () [inline, static]

The name : "sinh".

```
37                          { return HxString("sinh"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSinh.h**

8.445 HxUpoSqrt Class Template Reference

Pixel functor for computation of square root.

`#include <HxUpoSqrt.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoSqrt** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.sqrt() #.

Static Public Methods

- **HxString className** ()
  The name : "sqrt".

8.445.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSqrt< DstValT, SrcValT >

Pixel functor for computation of square root.

8.445.2 Member Typedef Documentation

8.445.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoSqrt::TransVarianceCategory

Functor is translation invariant.

8.445.3 Constructor & Destructor Documentation

8.445.3.1 template<class DstValT, class SrcValT> HxUpoSqrt< DstValT, SrcValT >::HxUpoSqrt (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.445.4 Member Function Documentation

8.445.4.1 template<class DstValT, class SrcValT> DstValT HxUpoSqrt< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sqrt() #.

```
33                          { return x.sqrt(); }
```

8.445.4.2 template<class DstValT, class SrcValT> HxString HxUpoSqrt< DstValT, SrcValT >::className () [inline, static]

The name : ”sqrt”.

```
37                          { return HxString("sqrt"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSqrt.h**

8.446 HxUpoSum Class Template Reference

Pixel functor for computation of unary sum.

`#include <HxUpoSum.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoSum** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.sum() #.

Static Public Methods

- **HxString className** ()
  The name : ”sum”.

8.446.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoSum< DstValT, SrcValT >

Pixel functor for computation of unary sum.

8.446.2 Member Typedef Documentation

8.446.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoSum::TransVarianceCategory

Functor is translation invariant.

8.446.3 Constructor & Destructor Documentation

8.446.3.1 template<class DstValT, class SrcValT> HxUpoSum< DstValT, SrcValT >::HxUpoSum (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.446.4 Member Function Documentation

8.446.4.1 template<class DstValT, class SrcValT> DstValT HxUpoSum< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.sum() #.

```
33                          { return x.sum(); }
```

8.446.4.2 template<class DstValT, class SrcValT> HxString HxUpoSum< DstValT, SrcValT >::className () [inline, static]

The name : ”sum”.

```
37                          { return HxString("sum"); }
```

The documentation for this class was generated from the following file:

- **HxUpoSum.h**

8.447 HxUpoTan Class Template Reference

Pixel functor for computation of tangent.

`#include <HxUpoTan.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoTan** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.tan() #.

Static Public Methods

- **HxString className** ()
  The name : "tan".

8.447.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoTan< DstValT, SrcValT >

Pixel functor for computation of tangent.

8.447.2 Member Typedef Documentation

8.447.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoTan::TransVarianceCategory

Functor is translation invariant.

8.447.3 Constructor & Destructor Documentation

8.447.3.1 template<class DstValT, class SrcValT> HxUpoTan< DstValT, SrcValT >::HxUpoTan (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.447.4 Member Function Documentation

8.447.4.1 template<class DstValT, class SrcValT> DstValT HxUpoTan< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.tan() #.

```
33                          { return x.tan(); }
```

8.447.4.2 template<class DstValT, class SrcValT> HxString HxUpoTan< DstValT, SrcValT >::className () [inline, static]

The name : ”tan”.

```
37                          { return HxString("tan"); }
```

The documentation for this class was generated from the following file:

- **HxUpoTan.h**

8.448 HxUpoTanh Class Template Reference

Pixel functor for computation of hyperbolic tangent.

`#include <HxUpoTanh.h>`

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **HxUpoTanh** (**HxTagList** &)
  Constructor : empty.
- DstValT **doIt** (const SrcValT &x)
  Actual operation : # return x.tanh() #.

Static Public Methods

- **HxString className** ()
  The name : ”tanh”.

8.448.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoTanh< DstValT, SrcValT >

Pixel functor for computation of hyperbolic tangent.

8.448.2 Member Typedef Documentation

8.448.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoTanh::TransVarianceCategory

Functor is translation invariant.

8.448.3 Constructor & Destructor Documentation

8.448.3.1 template<class DstValT, class SrcValT> HxUpoTanh< DstValT, SrcValT >::HxUpoTanh (HxTagList &) [inline]

Constructor : empty.

```
29                          {}
```

8.448.4 Member Function Documentation

8.448.4.1 template<class DstValT, class SrcValT> DstValT HxUpoTanh< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation : # return x.tanh() #.

```
33                          { return x.tanh(); }
```

8.448.4.2 template<class DstValT, class SrcValT> HxString HxUpoTanh< DstValT, SrcValT >::className () [inline, static]

The name : ”tanh”.

```
37                          { return HxString("tanh"); }
```

The documentation for this class was generated from the following file:

- **HxUpoTanh.h**

8.449 HxUpoTriStateThreshold Class Template Reference

Pixel functor for computation of tri state threshold.

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**

  Functor is translation invariant.

Public Methods

- **HxUpoTriStateThreshold** (**HxTagList** &)

  Constructor : get parameters from taglist.

- DstValT **doIt** (const SrcValT &x)

  Actual operation.

Static Public Methods

- **HxString className** ()

  The name : "triStateThreshold".

8.449.1 Detailed Description

template<class DstValT, class SrcValT> class HxUpoTriStateThreshold< DstValT, SrcValT >

Pixel functor for computation of tri state threshold.

8.449.2 Member Typedef Documentation

8.449.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar HxUpoTriStateThreshold::TransVarianceCategory

Functor is translation invariant.

8.449.3 Constructor & Destructor Documentation

8.449.3.1 template<class DstValT, class SrcValT> HxUpoTriStateThreshold< DstValT, SrcValT >::HxUpoTriStateThreshold (HxTagList & *tl*)

Constructor : get parameters from taglist.

```
43 {
44     _level = HxGetTag<HxValue>(tl, "level");
45     _v1 = HxGetTag<HxValue>(tl, "v1");
46     _v2 = HxGetTag<HxValue>(tl, "v2");
47     _v3 = HxGetTag<HxValue>(tl, "v3");
48 }
```

8.449.4 Member Function Documentation

8.449.4.1 template<class DstValT, class SrcValT> DstValT HxUpoTriStateThreshold< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation.

```
53 {
54     if (x < _level)
55         return _v1;
56     if (x > _level)
57         return _v3;
58     return _v2;
59 }
```

8.449.4.2 template<class DstValT, class SrcValT> HxString HxUpoTriStateThreshold< DstValT, SrcValT >::className () [static]

The name : ”triStateThreshold”.

```
64 {
65     return HxString("triStateThreshold");
66 }
```

The documentation for this class was generated from the following file:

- HxTriStateThreshold.c

8.450 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. 1164).
- **HxScalarDouble** & **HxScalarDoubleValue** ()
  Access value as **HxScalarDouble** (p. 1145).
- **HxVec2Int** & **HxVec2IntValue** ()
  Access value as **HxVec2Int** (p. 1281).
- **HxVec2Double** & **HxVec2DoubleValue** ()
  Access value as **HxVec2Double** (p. 1262).
- **HxVec3Int** & **HxVec3IntValue** ()
  Access value as **HxVec3Int** (p. 1321).
- **HxVec3Double** & **HxVec3DoubleValue** ()
  Access value as **HxVec3Double** (p. 1301).
- **HxComplex** & **HxComplexValue** ()
  Access value as **HxComplex** (p. 506).

Conversion

- **operator HxScalarInt** () const
  Cast to **HxScalarInt** (p. 1164).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 1145).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec2Double** () const

Cast to **HxVec2Double** (p. 1262).

- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

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)

8.450.1 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.

8.450.2 Member Enumeration Documentation

8.450.2.1 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 };
```

8.450.3 Constructor & Destructor Documentation

8.450.3.1 HxValue::HxValue (int *v* = 0) [inline]

Construction from native integer.

```
159 {
160     _tag._tag = SI;
161     new(&_val[0]) HxScalarInt(v);
162 }
```

8.450.3.2 HxValue::HxValue (double *v*) [inline]

Construction from native double.

```
166 {
167     _tag._tag = SD;
168     new(&_val[0]) HxScalarDouble(v);
169 }
```

8.450.3.3 HxValue::HxValue (HxScalarInt *v*) [inline]

Construction from scalar integer.

```
173 {
174     _tag._tag = SI;
175     new(&_val[0]) HxScalarInt(v);
176 }
```

8.450.3.4 HxValue::HxValue (HxScalarDouble *v*) [inline]

Construction from scalar double.

```
180 {
181     _tag._tag = SD;
182     new(&_val[0]) HxScalarDouble(v);
183 }
```

8.450.3.5 HxValue::HxValue (HxVec2Int *v*) [inline]

Construction from vector of 2 integers.

```
187 {
188     _tag._tag = V2I;
189     new(&_val[0]) HxVec2Int(v);
190 }
```

8.450.3.6 HxValue::HxValue (HxVec2Double *v*) [inline]

Construction from vector of 2 doubles.

```
194 {
195     _tag._tag = V2D;
196     new(&_val[0]) HxVec2Double(v);
197 }
```

8.450.3.7 HxValue::HxValue (HxVec3Int *v*) [inline]

Construction from vector of 3 integers.

```
201 {
202     _tag._tag = V3I;
203     new(&_val[0]) HxVec3Int(v);
204 }
```

8.450.3.8 HxValue::HxValue (HxVec3Double *v*) [inline]

Construction from vector of 3 doubles.

```
208 {
209     _tag._tag = V3D;
210     new(&_val[0]) HxVec3Double(v);
211 }
```

8.450.3.9 HxValue::HxValue (HxComplex *v*) [inline]

Construction from complex.

```
215 {
216     _tag._tag = CPL;
217     new(&_val[0]) HxComplex(v);
218 }
```

8.450.4 Member Function Documentation

8.450.4.1 HxValue::Tag HxValue::tag () const [inline]

The type of the value stored.

```
222 {
223     return _tag._tag;
224 }
```

8.450.4.2 HxScalarInt & HxValue::HxScalarIntValue () [inline]

Access value as **HxScalarInt** (p. 1164).

```
228 {
229     check(SI);
230     return *(HxScalarInt *) &_val[0];
231 }
```

8.450.4.3 HxScalarDouble & HxValue::HxScalarDoubleValue () [inline]

Access value as **HxScalarDouble** (p. 1145).

```
235 {
236     check(SD);
237     return *(HxScalarDouble *) &_val[0];
238 }
```

8.450.4.4 HxVec2Int & HxValue::HxVec2IntValue () [inline]

Access value as **HxVec2Int** (p. 1281).

```
242 {
243     check(V2I);
244     return *(HxVec2Int *) &_val[0];
245 }
```

8.450.4.5 HxVec2Double & HxValue::HxVec2DoubleValue () [inline]

Access value as **HxVec2Double** (p. 1262).

```
249 {
250     check(V2D);
251     return *(HxVec2Double *) &_val[0];
252 }
```

8.450.4.6 HxVec3Int & HxValue::HxVec3IntValue () [inline]

Access value as **HxVec3Int** (p. 1321).

```
256 {
257     check(V3I);
258     return *(HxVec3Int *) &_val[0];
259 }
```

8.450.4.7 HxVec3Double & HxValue::HxVec3DoubleValue () [inline]

Access value as **HxVec3Double** (p. 1301).

```
263 {
264     check(V3D);
265     return *(HxVec3Double *) &_val[0];
266 }
```

8.450.4.8 HxComplex & HxValue::HxComplexValue () [inline]

Access value as **HxComplex** (p. 506).

```
270 {
271     check(CPL);
272     return *(HxComplex *) &_val[0];
273 }
```

8.450.4.9 HxValue::operator HxScalarInt () const [inline]

Cast to **HxScalarInt** (p. 1164).

```
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 }
```

8.450.4.10 HxValue::operator HxScalarDouble () const [inline]

Cast to **HxScalarDouble** (p. 1145).

```
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 }
```

8.450.4.11 HxValue::operator HxVec2Int () const [inline]

Cast to **HxVec2Int** (p. 1281).

```
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 }
```

8.450.4.12 HxValue::operator HxVec2Double () const [inline]

Cast to **HxVec2Double** (p. 1262).

```
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 }
```

8.450.4.13 HxValue::operator HxVec3Int () const [inline]

Cast to **HxVec3Int** (p. 1321).

```
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 }
```

8.450.4.14 HxValue::operator HxVec3Double () const [inline]

Cast to **HxVec3Double** (p. 1301).

```
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 }
```

8.450.4.15 HxValue::operator HxComplex () const [inline]

Cast to **HxComplex** (p. 506).

```
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

8.451 HxValueList Class Reference

Class definition for list of **HxValue** (p. 1253)'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.

8.451.1 Detailed Description

Class definition for list of **HxValue** (p. 1253)'s.

Specialization of list from STL.

8.451.2 Member Typedef Documentation

8.451.2.1 typedef std::back_insert_iterator<HxValueList> HxValueList::back_insert_iterator

back inserter.

8.451.3 Member Function Documentation

8.451.3.1 HxValueList & HxValueList::operator<< (const HxValue & *s*) [inline]

Add value to the list.

```
48 {
49     push_back(s);
50     return *this;
51 }
```

8.451.3.2 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**

8.452 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. 1164).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 1145).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.452.1 Detailed Description

Class definition vector of 2 doubles.

8.452.2 Constructor & Destructor Documentation

8.452.2.1 HxVec2Double::HxVec2Double () [inline]

Default constructor.

```
323 {
324 }
```

8.452.2.2 HxVec2Double::HxVec2Double (double *x*, double *y*) [inline]

Conversion from native type.

```
328 {
329     _values[0] = x;
330     _values[1] = y;
331 }
```

8.452.2.3 HxVec2Double::HxVec2Double (const HxVec2Double & *v*) [inline]

Copy constructor.

```
335 {
336     _values[0] = v._values[0];
337     _values[1] = v._values[1];
338 }
```

8.452.3 Member Function Documentation

8.452.3.1 int HxVec2Double::dim () const [inline]

Dimensionality.

```
356 {
357     return 2;
358 }
```

8.452.3.2 double HxVec2Double::x () const [inline]

Value of first element.

```
362 {
363     return _values[0];
364 }
```

8.452.3.3 double HxVec2Double::y () const [inline]

Value of second element.

```
368 {
369     return _values[1];
370 }
```

8.452.3.4 double HxVec2Double::getValue (int *dim*) const [inline]

Element in given dimension.

```
374 {
375     return _values[dim - 1];
376 }
```

8.452.3.5 HxVec2Double::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 1164).

```
26 {
27     return (int) _values[0];
28 }
```

8.452.3.6 HxVec2Double::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 1145).

```
31 {
32     return _values[0];
33 }
```

8.452.3.7 HxVec2Double::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 1281).

```
36 {
37     return HxVec2Int(int(_values[0]), int(_values[1]));
38 }
```

8.452.3.8 HxVec2Double::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 1321).

```
41 {
42     return HxVec3Int(int(_values[0]), int(_values[1]), 0);
43 }
```

8.452.3.9 HxVec2Double::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 1301).

```
46 {
47     return HxVec3Double(_values[0], _values[1], 0);
48 }
```

8.452.3.10 HxVec2Double::operator HxComplex () const

Cast to **HxComplex** (p. 506).

```
51 {
52     return HxComplex(_values[0], _values[1]);
53 }
```

8.452.3.11 int HxVec2Double::operator== (const HxVec2Double & *v*) const [inline]

Equal.

```
386 {
387     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]);
388 }
```

8.452.3.12 int HxVec2Double::operator!= (const HxVec2Double & *v*) const [inline]

Not equal.

```
392 {
393     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]);
394 }
```

8.452.3.13 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 }
```

8.452.3.14 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 }
```

8.452.3.15 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 }
```

8.452.3.16 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 }
```

8.452.3.17 HxVec2Double HxVec2Double::operator- () const [inline]

Negation.

```
426 {
427     return HxVec2Double(-_values[0], -_values[1]);
428 }
```

8.452.3.18 HxVec2Double HxVec2Double::complement () const [inline]

Complement.

```
432 {
433     return HxVec2Double(-_values[0], -_values[1]);
434 }
```

8.452.3.19 HxVec2Double HxVec2Double::abs () const [inline]

Absolute value.

```
438 {
439     return HxVec2Double(fabs(_values[0]), fabs(_values[1]));
440 }
```

8.452.3.20 HxVec2Double HxVec2Double::ceil () const [inline]

Ceiling.

```
444 {
445     return HxVec2Double(::ceil(_values[0]), ::ceil(_values[1]));
446 }
```

8.452.3.21 HxVec2Double HxVec2Double::floor () const [inline]

Floor.

```
450 {
451     return HxVec2Double(::floor(_values[0]), ::floor(_values[1]));
452 }
```

8.452.3.22 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 }
```

8.452.3.23 HxScalarDouble HxVec2Double::sum () const [inline]

Sum.

```
710 {
711     return _values[0] + _values[1];
712 }
```

8.452.3.24 HxScalarDouble HxVec2Double::product () const [inline]

Product.

```
716 {
717     return _values[0] * _values[1];
718 }
```

8.452.3.25 HxScalarDouble HxVec2Double::min () const [inline]

Minimum.

```
722 {
723     return (_values[0] < _values[1]) ? _values[0] : _values[1];
724 }
```

8.452.3.26 HxScalarDouble HxVec2Double::max () const [inline]

Maximum.

```
728 {
729     return (_values[0] > _values[1]) ? _values[0] : _values[1];
730 }
```

8.452.3.27 HxScalarDouble HxVec2Double::norm1 () const

L1 norm.

```
57 {
58     return fabs(_values[0]) + fabs(_values[1]);
59 }
```

8.452.3.28 HxScalarDouble HxVec2Double::norm2 () const

L2 norm.

```
63 {
64     return ::sqrt(_values[0]*_values[0] + _values[1]*_values[1]);
65 }
```

8.452.3.29 HxScalarDouble HxVec2Double::normInf () const

L infinity norm.

```
69 {
70     return (fabs(_values[0]) > fabs(_values[1])) ? fabs(_values[0]) :
71                                                    fabs(_values[1]);
72 }
```

8.452.3.30 HxVec2Double HxVec2Double::sqrt () const [inline]

Square root.

```
463 {
464     return HxVec2Double(::sqrt(_values[0]), ::sqrt(_values[1]));
465 }
```

8.452.3.31 HxVec2Double HxVec2Double::sin () const [inline]

Sine.

```
469 {
470     return HxVec2Double(::sin(_values[0]), ::sin(_values[1]));
471 }
```

8.452.3.32 HxVec2Double HxVec2Double::cos () const [inline]

Cosine.

```
475 {
476     return HxVec2Double(::cos(_values[0]), ::cos(_values[1]));
477 }
```

8.452.3.33 HxVec2Double HxVec2Double::tan () const [inline]

Tangent.

```
481 {
482     return HxVec2Double(::tan(_values[0]), ::tan(_values[1]));
483 }
```

8.452.3.34 HxVec2Double HxVec2Double::asin () const [inline]

Arc sine.

```
487 {
488     return HxVec2Double(::asin(_values[0]), ::asin(_values[1]));
489 }
```

8.452.3.35 HxVec2Double HxVec2Double::acos () const [inline]

Arc cosine.

```
493 {
494     return HxVec2Double(::acos(_values[0]), ::acos(_values[1]));
495 }
```

8.452.3.36 HxVec2Double HxVec2Double::atan () const [inline]

Arc tangent.

```
499 {
500     return HxVec2Double(::atan(_values[0]), ::atan(_values[1]));
501 }
```

8.452.3.37 HxScalarDouble HxVec2Double::atan2 () const

Arc tangent.

```
76 {
77     return HxScalarDouble(::atan2(_values[0], _values[1]));
78 }
```

8.452.3.38 HxVec2Double HxVec2Double::sinh () const [inline]

Hyperbolic sine.

```
505 {
506     return HxVec2Double(::sinh(_values[0]), ::sinh(_values[1]));
507 }
```

8.452.3.39 HxVec2Double HxVec2Double::cosh () const [inline]

Hyperbolic cosine.

```
511 {
512     return HxVec2Double(::cosh(_values[0]), ::cosh(_values[1]));
513 }
```

8.452.3.40 HxVec2Double HxVec2Double::tanh () const [inline]

Hyperbolic tangent.

```
517 {
518     return HxVec2Double(::tanh(_values[0]), ::tanh(_values[1]));
519 }
```

8.452.3.41 HxVec2Double HxVec2Double::exp () const [inline]

Exponent.

```
523 {
524     return HxVec2Double(::exp(_values[0]), ::exp(_values[1]));
525 }
```

8.452.3.42 HxVec2Double HxVec2Double::log () const [inline]

Natural logarithm.

```
529 {
530     return HxVec2Double(::log(_values[0]), ::log(_values[1]));
531 }
```

8.452.3.43 HxVec2Double HxVec2Double::log10 () const [inline]

Base 10 logarithm.

```
535 {
536     return HxVec2Double(::log10(_values[0]), ::log10(_values[1]));
537 }
```

8.452.3.44 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 }
```

8.452.3.45 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 }
```

8.452.3.46 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 }
```

8.452.3.47 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 }
```

8.452.3.48 HxVec2Double HxVec2Double::min (const HxVec2Double & *v*) const [inline]

Minimum.

```
601 {
602     return (operator<(v)) ? (*this) : v;
603 }
```

8.452.3.49 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 }
```

8.452.3.50 HxVec2Double HxVec2Double::max (const HxVec2Double & *v*) const [inline]

Maximum.

```
616 {
617     return (operator>(v)) ? (*this) : v;
618 }
```

8.452.3.51 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 }
```

8.452.3.52 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 }
```

8.452.3.53 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 }
```

8.452.3.54 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 }
```

8.452.3.55 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 }
```

8.452.3.56 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 }
```

8.452.3.57 HxVec2Double HxVec2Double::mod (const HxVec2Double & *v*) const [inline]

Modulo.

```
668 {
669     return (*this);
670 }
```

8.452.3.58 HxVec2Double HxVec2Double::and (const HxVec2Double & *v*) const [inline]

And.

```
674 {
675     return (*this);
676 }
```

8.452.3.59 HxVec2Double HxVec2Double::or (const HxVec2Double & *v*) const [inline]

Or.

```
680 {
681     return (*this);
682 }
```

8.452.3.60 HxVec2Double HxVec2Double::xor (const HxVec2Double & *v*) const [inline]

Xor.

```
686 {
687     return (*this);
688 }
```

8.452.3.61 HxVec2Double HxVec2Double::leftShift (const HxVec2Double & *v*) const [inline]

Left shift.

```
692 {
693     return (*this);
694 }
```

8.452.3.62 HxVec2Double HxVec2Double::rightShift (const HxVec2Double & *v*) const [inline]

Right shift.

```
698 {
699     return (*this);
700 }
```

8.452.3.63 HxScalarDouble HxVec2Double::dot (const HxVec2Double & *v*) const

Dot product.

```
82 {
83     return (_values[0] * v._values[0]) + (_values[1] * v._values[1]);
84 }
```

8.452.3.64 HxVec2Double HxVec2Double::cross (const HxVec2Double & *v*) const [inline]

Cross product.

```
704 {
705     return HxVec2Double(0, 0);
706 }
```

8.452.3.65 STD_OSTREAM & HxVec2Double::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
88 {
89     return os << "(" << _values[0] << "," << _values[1] << ")";
90 }
```

8.452.3.66 HxString HxVec2Double::toString () const

Value as a string.

```
93                              {
94     return HxString("(") + makeString(_values[0]) + ", "
95             + makeString(_values[1]) + ")";
96 }
```

8.452.4 Friends And Related Function Documentation

8.452.4.1 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 }
```

8.452.4.2 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 }
```

8.452.4.3 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 }
```

8.452.4.4 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 }
```

8.452.5 Member Data Documentation

8.452.5.1 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.

8.452.5.2 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

8.453 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. 1164).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 1145).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 1262).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.453.1 Detailed Description

Class definition vector of 2 integers.

8.453.2 Constructor & Destructor Documentation

8.453.2.1 HxVec2Int::HxVec2Int () [inline]

Default constructor.

```
319 {
320 }
```

8.453.2.2 HxVec2Int::HxVec2Int (int *x*, int *y*) [inline]

Conversion from native type.

```
324 {
325     _values[0] = x;
326     _values[1] = y;
327 }
```

8.453.2.3 HxVec2Int::HxVec2Int (const HxVec2Int & *v*) [inline]

Copy constructor.

```
331 {
332     _values[0] = v._values[0];
333     _values[1] = v._values[1];
334 }
```

8.453.3 Member Function Documentation

8.453.3.1 int HxVec2Int::dim () const [inline]

Dimensionality.

```
344 {
345     return 2;
346 }
```

8.453.3.2 int HxVec2Int::x () const [inline]

Value of first element.

```
350 {
351     return _values[0];
352 }
```

8.453.3.3 int HxVec2Int::y () const [inline]

Value of second element.

```
356 {
357     return _values[1];
358 }
```

8.453.3.4 int HxVec2Int::getValue (int *dim*) const [inline]

Element in given dimension.

```
362 {
363     return _values[dim - 1];
364 }
```

8.453.3.5 HxVec2Int::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 1164).

```
26 {
27     return _values[0];
28 }
```

8.453.3.6 HxVec2Int::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 1145).

```
31 {
32     return (double) _values[0];
33 }
```

8.453.3.7 HxVec2Int::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 1262).

```
36 {
37     return HxVec2Double(_values[0], _values[1]);
38 }
```

8.453.3.8 HxVec2Int::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 1321).

```
41 {
42     return HxVec3Int(_values[0], _values[1], 0);
43 }
```

8.453.3.9 HxVec2Int::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 1301).

```
46 {
47     return HxVec3Double(_values[0], _values[1], 0);
48 }
```

8.453.3.10 HxVec2Int::operator HxComplex () const

Cast to **HxComplex** (p. 506).

```
51 {
52     return HxComplex(_values[0], _values[1]);
53 }
```

8.453.3.11 int HxVec2Int::operator== (const HxVec2Int & *v*) const [inline]

Equal.

```
374 {
375     return (_values[0] == v._values[0]) && (_values[1] == v._values[1]);
376 }
```

8.453.3.12 int HxVec2Int::operator!= (const HxVec2Int & *v*) const [inline]

Not equal.

```
380 {
381     return (_values[0] != v._values[0]) || (_values[1] != v._values[1]);
382 }
```

8.453.3.13 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 }
```

8.453.3.14 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 }
```

8.453.3.15 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 }
```

8.453.3.16 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 }
```

8.453.3.17 HxVec2Int HxVec2Int::operator- () const [inline]

Negation.

```
414 {
415     return HxVec2Int(-_values[0], -_values[1]);
416 }
```

8.453.3.18 HxVec2Int HxVec2Int::complement () const [inline]

Complement.

```
420 {
421     return HxVec2Int(~_values[0], ~_values[1]);
422 }
```

8.453.3.19 HxVec2Int HxVec2Int::abs () const [inline]

Absolute value.

```
426 {
427     return HxVec2Int(::abs(_values[0]), ::abs(_values[1]));
428 }
```

8.453.3.20 HxVec2Int HxVec2Int::ceil () const [inline]

Ceiling.

```
432 {
433     return *this;
434 }
```

8.453.3.21 HxVec2Int HxVec2Int::floor () const [inline]

Floor.

```
438 {
439     return *this;
440 }
```

8.453.3.22 HxVec2Int HxVec2Int::round () const [inline]

Round.

```
444 {
445     return *this;
446 }
```

8.453.3.23 HxScalarInt HxVec2Int::sum () const [inline]

Sum.

```
625 {
626     return _values[0] + _values[1];
627 }
```

8.453.3.24 HxScalarInt HxVec2Int::product () const [inline]

Product.

```
631 {
632     return _values[0] * _values[1];
633 }
```

8.453.3.25 HxScalarInt HxVec2Int::min () const [inline]

Minimum.

```
637 {
638     return (_values[0] < _values[1]) ? _values[0] : _values[1];
639 }
```

8.453.3.26 HxScalarInt HxVec2Int::max () const [inline]

Maximum.

```
643 {
644     return (_values[0] > _values[1]) ? _values[0] : _values[1];
645 }
```

8.453.3.27 HxScalarInt HxVec2Int::norm1 () const

L1 norm.

```
57 {
58     return ::abs(_values[0]) + ::abs(_values[1]);
59 }
```

8.453.3.28 HxScalarDouble HxVec2Int::norm2 () const

L2 norm.

```
63 {
64     return ::sqrt(double(_values[0])*_values[0] + double(_values[1])*_values[1]);
65 }
```

8.453.3.29 HxScalarInt HxVec2Int::normInf () const

L infinity norm.

```
69 {
70     return (::abs(_values[0]) > ::abs(_values[1])) ? ::abs(_values[0]) :
71                                                      ::abs(_values[1]);
72 }
```

8.453.3.30 HxVec2Double HxVec2Int::sqrt () const

Square root.

```
76 {
77     return HxVec2Double(::sqrt(double(_values[0])), ::sqrt(double(_values[1])));
78 }
```

8.453.3.31 HxVec2Double HxVec2Int::sin () const

Sine.

```
82 {
83     return HxVec2Double(::sin(double(_values[0])), ::sin(double(_values[1])));
84 }
```

8.453.3.32 HxVec2Double HxVec2Int::cos () const

Cosine.

```
88 {
89     return HxVec2Double(::cos(double(_values[0])), ::cos(double(_values[1])));
90 }
```

8.453.3.33 HxVec2Double HxVec2Int::tan () const

Tangent.

```
94 {
95     return HxVec2Double(::tan(double(_values[0])), ::tan(double(_values[1])));
96 }
```

8.453.3.34 HxVec2Double HxVec2Int::asin () const

Arc sine.

```
100 {
101     return HxVec2Double(::asin(double(_values[0])), ::asin(double(_values[1])));
102 }
```

8.453.3.35 HxVec2Double HxVec2Int::acos () const

Arc cosine.

```
106 {
107     return HxVec2Double(::acos(double(_values[0])), ::acos(double(_values[1])));
108 }
```

8.453.3.36 HxVec2Double HxVec2Int::atan () const

Arc tangent.

```
112 {
113     return HxVec2Double(::atan(double(_values[0])), ::atan(double(_values[1])));
114 }
```

8.453.3.37 HxScalarDouble HxVec2Int::atan2 () const

Arc tangent.

```
118 {
119     return HxScalarDouble(::atan2(double(_values[0]), double(_values[1])));
120 }
```

8.453.3.38 HxVec2Double HxVec2Int::sinh () const

Hyperbolic sine.

```
124 {
125     return HxVec2Double(::sinh(double(_values[0])), ::sinh(double(_values[1])));
126 }
```

8.453.3.39 HxVec2Double HxVec2Int::cosh () const

Hyperbolic cosine.

```
130 {
131     return HxVec2Double(::cosh(double(_values[0])), ::cosh(double(_values[1])));
132 }
```

8.453.3.40 HxVec2Double HxVec2Int::tanh () const

Hyperbolic tangent.

```
136 {
137     return HxVec2Double(::tanh(double(_values[0])), ::tanh(double(_values[1])));
138 }
```

8.453.3.41 HxVec2Double HxVec2Int::exp () const

Exponent.

```
142 {
143     return HxVec2Double(::exp(double(_values[0])), ::exp(double(_values[1])));
144 }
```

8.453.3.42 HxVec2Double HxVec2Int::log () const

Natural logarithm.

```
148 {
149     return HxVec2Double(::log(double(_values[0])), ::log(double(_values[1])));
150 }
```

8.453.3.43 HxVec2Double HxVec2Int::log10 () const

Base 10 logarithm.

```
154 {
155     return HxVec2Double(::log10(double(_values[0])), ::log10(double(_values[1])));
156 }
```

8.453.3.44 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 }
```

8.453.3.45 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 }
```

8.453.3.46 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 }
```

8.453.3.47 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 }
```

8.453.3.48 HxVec2Int HxVec2Int::min (const HxVec2Int & *v*) const [inline]

Minimum.

```
510 {
511     return (operator<(v)) ? (*this) : v;
512 }
```

8.453.3.49 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 }
```

8.453.3.50 HxVec2Int HxVec2Int::max (const HxVec2Int & *v*) const [inline]

Maximum.

```
525 {
526     return (operator>(v)) ? (*this) : v;
527 }
```

8.453.3.51 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 }
```

8.453.3.52 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 }
```

8.453.3.53 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 }
```

8.453.3.54 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 }
```

8.453.3.55 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 }
```

8.453.3.56 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 }
```

8.453.3.57 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 }
```

8.453.3.58 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 }
```

8.453.3.59 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 }
```

8.453.3.60 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 }
```

8.453.3.61 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 }
```

8.453.3.62 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 }
```

8.453.3.63 HxScalarInt HxVec2Int::dot (const HxVec2Int & *v*) const

Dot product.

```
160 {
161     return (_values[0] * v._values[0]) + (_values[1] * v._values[1]);
162 }
```

8.453.3.64 HxVec2Int HxVec2Int::cross (const HxVec2Int & *v*) const [inline]

Cross product.

```
619 {
620     return HxVec2Int(0, 0);
621 }
```

8.453.3.65 STD_OSTREAM & HxVec2Int::put (STD_OSTREAM & *os*) const

Print value on stream.

For global operator<<

```
166 {
167     return os << "(" << _values[0] << "," << _values[1] << ")";
168 }
```

8.453.3.66 HxString HxVec2Int::toString () const

Value as a string.

```
171                             {
172     return HxString("(") + makeString(_values[0]) + ", "
173             + makeString(_values[1]) + ")";
174 }
```

8.453.4 Friends And Related Function Documentation

8.453.4.1 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 }
```

8.453.4.2 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 }
```

8.453.4.3 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 }
```

8.453.4.4 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 }
```

8.453.5 Member Data Documentation

8.453.5.1 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.

8.453.5.2 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

8.454 HxVec2Tem Class Template Reference

Template class for representation of vector of 2 scalars of type T.

`#include <HxVec2Tem.h>`

Public Methods

- **HxVec2Tem** ()
- **HxVec2Tem** (T x, T y)
- **HxVec2Tem** (const HxVec2Tem< T > &v)
- **HxVec2Tem** (int v)
- **HxVec2Tem** (double v)
- **HxVec2Tem** (const **HxVec2Int** &v)
- **HxVec2Tem** (const **HxVec2Double** &v)
- void * **operator new** (size_t, void *=0)
- T **x** () const
- T **y** () const
- T & **value** (int dimension)
- **operator int** () const
- **operator double** () const
- **operator HxVec2Int** () const
- **operator HxVec2Double** () const
- HxVec2Tem< T > & **operator+=** (const HxVec2Tem< T > &v)
- HxVec2Tem< T > & **operator-=** (const HxVec2Tem< T > &v)
- HxVec2Tem< T > & **operator** *= (const HxVec2Tem< T > &v)
- HxVec2Tem< T > & **operator/=** (const HxVec2Tem< T > &v)
- STD_OSTREAM & **put** (STD_OSTREAM &os) const

Friends

- HxVec2Tem< T > **operator+** (const HxVec2Tem< T > &v1, const HxVec2Tem< T > &v2)
- HxVec2Tem< T > **operator-** (const HxVec2Tem< T > &v1, const HxVec2Tem< T > &v2)
- HxVec2Tem< T > **operator** * (const HxVec2Tem< T > &v1, const HxVec2Tem< T > &v2)
- HxVec2Tem< T > **operator/** (const HxVec2Tem< T > &v1, const HxVec2Tem< T > &v2)

8.454.1 Detailed Description

template<class T> class HxVec2Tem< T >

Template class for representation of vector of 2 scalars of type T.

The documentation for this class was generated from the following files:

- **HxVec2Tem.h**
- HxVec2Tem.c

8.455 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. 1164).
- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 1145).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 1262).
- **operator HxVec3Int** () const
  Cast to **HxVec3Int** (p. 1321).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.455.1 Detailed Description

Class definition vector of 3 doubles.

8.455.2 Constructor & Destructor Documentation

8.455.2.1 HxVec3Double::HxVec3Double () [inline]

Default constructor.

```
323 {
324 }
```

8.455.2.2 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 }
```

8.455.2.3 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 }
```

8.455.3 Member Function Documentation

8.455.3.1 int HxVec3Double::dim () const [inline]

Dimensionality.

```
350 {
351     return 3;
352 }
```

8.455.3.2 double HxVec3Double::x () const [inline]

Value of first element.

```
356 {
357     return _values[0];
358 }
```

8.455.3.3 double HxVec3Double::y () const [inline]

Value of second element.

```
362 {
363     return _values[1];
364 }
```

8.455.3.4 double HxVec3Double::z () const [inline]

Value of third element.

```
368 {
369     return _values[2];
370 }
```

8.455.3.5 double HxVec3Double::getValue (int *dimension*) const [inline]

Element in given dimension.

```
374 {
375     return _values[dimension - 1];
376 }
```

8.455.3.6 HxVec3Double::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 1164).

```
28 {
29     return (int) _values[0];
30 }
```

8.455.3.7 HxVec3Double::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 1145).

```
33 {
34     return _values[0];
35 }
```

8.455.3.8 HxVec3Double::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 1281).

```
38 {
39     return HxVec2Int(int(_values[0]), int(_values[1]));
40 }
```

8.455.3.9 HxVec3Double::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 1262).

```
43 {
44     return HxVec2Double(_values[0], _values[1]);
45 }
```

8.455.3.10 HxVec3Double::operator HxVec3Int () const

Cast to **HxVec3Int** (p. 1321).

```
48 {
49     return HxVec3Int(int(_values[0]), int(_values[1]), int(_values[2]));
50 }
```

8.455.3.11 HxVec3Double::operator HxComplex () const

Cast to **HxComplex** (p. 506).

```
53 {
54     return HxComplex(_values[0], _values[1]);
55 }
```

8.455.3.12 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 }
```

8.455.3.13 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 }
```

8.455.3.14 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 }
```

8.455.3.15 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 }
```

8.455.3.16 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 }
```

8.455.3.17 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 }
```

8.455.3.18 HxVec3Double HxVec3Double::operator- () const [inline]

Negation.

```
428 {
429     return HxVec3Double(-_values[0], -_values[1], -_values[2]);
430 }
```

8.455.3.19 HxVec3Double HxVec3Double::complement () const [inline]

Complement.

```
434 {
435     return HxVec3Double(-_values[0], -_values[1], -_values[2]);
436 }
```

8.455.3.20 HxVec3Double HxVec3Double::abs () const [inline]

Absolute value.

```
440 {
441     return HxVec3Double(fabs(_values[0]), fabs(_values[1]), fabs(_values[2]));
442 }
```

8.455.3.21 HxVec3Double HxVec3Double::ceil () const [inline]

Ceiling.

```
446 {
447     return HxVec3Double(::ceil(_values[0]),
448                         ::ceil(_values[1]),
449                         ::ceil(_values[2]));
450 }
```

8.455.3.22 HxVec3Double HxVec3Double::floor () const [inline]

Floor.

```
454 {
455     return HxVec3Double(::floor(_values[0]),
456                         ::floor(_values[1]),
457                         ::floor(_values[2]));
458 }
```

8.455.3.23 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 }
```

8.455.3.24 HxScalarDouble HxVec3Double::sum () const [inline]

Sum.

```
470 {
471     return _values[0] + _values[1] + _values[2];
472 }
```

8.455.3.25 HxScalarDouble HxVec3Double::product () const [inline]

Product.

```
476 {
477     return _values[0] * _values[1] * _values[2];
478 }
```

8.455.3.26 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 }
```

8.455.3.27 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 }
```

8.455.3.28 HxScalarDouble HxVec3Double::norm1 () const

L1 norm.

```
75 {
76     return fabs(_values[0]) + fabs(_values[1]) + fabs(_values[2]);
77 }
```

8.455.3.29 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 }
```

8.455.3.30 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 }
```

8.455.3.31 HxVec3Double HxVec3Double::sqrt () const [inline]

Square root.

```
482 {
483     return HxVec3Double(::sqrt(_values[0]),
484                         ::sqrt(_values[1]),
485                         ::sqrt(_values[2]));
486 }
```

8.455.3.32 HxVec3Double HxVec3Double::sin () const [inline]

Sine.

```
490 {
491     return HxVec3Double(::sin(_values[0]),
492                         ::sin(_values[1]),
493                         ::sin(_values[2]));
494 }
```

8.455.3.33 HxVec3Double HxVec3Double::cos () const [inline]

Cosine.

```
498 {
499     return HxVec3Double(::cos(_values[0]),
500                         ::cos(_values[1]),
501                         ::cos(_values[2]));
502 }
```

8.455.3.34 HxVec3Double HxVec3Double::tan () const [inline]

Tangent.

```
506 {
507     return HxVec3Double(::tan(_values[0]),
508                         ::tan(_values[1]),
509                         ::tan(_values[2]));
510 }
```

8.455.3.35 HxVec3Double HxVec3Double::asin () const [inline]

Arc sine.

```
514 {
515     return HxVec3Double(::asin(_values[0]),
516                         ::asin(_values[1]),
517                         ::asin(_values[2]));
518 }
```

8.455.3.36 HxVec3Double HxVec3Double::acos () const [inline]

Arc cosine.

```
522 {
523     return HxVec3Double(::acos(_values[0]),
524                         ::acos(_values[1]),
525                         ::acos(_values[2]));
526 }
```

8.455.3.37 HxVec3Double HxVec3Double::atan () const [inline]

Arc tangent.

```
530 {
531     return HxVec3Double(::atan(_values[0]),
532                         ::atan(_values[1]),
533                         ::atan(_values[2]));
534 }
```

8.455.3.38 HxScalarDouble HxVec3Double::atan2 () const

Arc tangent.

```
99 {
100     return HxScalarDouble(::atan2(_values[0], _values[1]));
101 }
```

8.455.3.39 HxVec3Double HxVec3Double::sinh () const [inline]

Hyperbolic sine.

```
538 {
539     return HxVec3Double(::sinh(_values[0]),
540                         ::sinh(_values[1]),
541                         ::sinh(_values[2]));
542 }
```

8.455.3.40 HxVec3Double HxVec3Double::cosh () const [inline]

Hyperbolic cosine.

```
546 {
547     return HxVec3Double(::cosh(_values[0]),
548                         ::cosh(_values[1]),
549                         ::cosh(_values[2]));
550 }
```

8.455.3.41 HxVec3Double HxVec3Double::tanh () const [inline]

Hyperbolic tangent.

```
554 {
555     return HxVec3Double(::tanh(_values[0]),
556                         ::tanh(_values[1]),
557                         ::tanh(_values[2]));
558 }
```

8.455.3.42 HxVec3Double HxVec3Double::exp () const [inline]

Exponent.

```
562 {
563     return HxVec3Double(::exp(_values[0]),
564                         ::exp(_values[1]),
565                         ::exp(_values[2]));
566 }
```

8.455.3.43 HxVec3Double HxVec3Double::log () const [inline]

Natural logarithm.

```
570 {
571     return HxVec3Double(::log(_values[0]),
572                         ::log(_values[1]),
573                         ::log(_values[2]));
574 }
```

8.455.3.44 HxVec3Double HxVec3Double::log10 () const [inline]

Base 10 logarithm.

```
578 {
579     return HxVec3Double(::log10(_values[0]),
580                         ::log10(_values[1]),
581                         ::log10(_values[2]));
582 }
```

8.455.3.45 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 }
```

8.455.3.46 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 }
```

8.455.3.47 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 }
```

8.455.3.48 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 }
```

8.455.3.49 HxVec3Double HxVec3Double::min (const HxVec3Double & *v*) const [inline]

Minimum.

```
654 {
655     return (operator<(v)) ? (*this) : v;
656 }
```

8.455.3.50 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 }
```

8.455.3.51 HxVec3Double HxVec3Double::max (const HxVec3Double & *v*) const [inline]

Maximum.

```
669 {
670     return (operator>(v)) ? (*this) : v;
671 }
```

8.455.3.52 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 }
```

8.455.3.53 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 }
```

8.455.3.54 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 }
```

8.455.3.55 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 }
```

8.455.3.56 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 }
```

8.455.3.57 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 }
```

8.455.3.58 HxVec3Double HxVec3Double::mod (const HxVec3Double & *v*) const [inline]

Modulo.

```
726 {
727     return (*this);
728 }
```

8.455.3.59 HxVec3Double HxVec3Double::and (const HxVec3Double & *v*) const [inline]

And.

```
732 {
733     return (*this);
734 }
```

8.455.3.60 HxVec3Double HxVec3Double::or (const HxVec3Double & *v*) const [inline]

Or.

```
738 {
739     return (*this);
740 }
```

8.455.3.61 HxVec3Double HxVec3Double::xor (const HxVec3Double & *v*) const [inline]

Xor.

```
744 {
745     return (*this);
746 }
```

8.455.3.62 HxVec3Double HxVec3Double::leftShift (const HxVec3Double & *v*) const [inline]

Left shift.

```
750 {
751     return (*this);
752 }
```

8.455.3.63 HxVec3Double HxVec3Double::rightShift (const HxVec3Double & *v*) const [inline]

Right shift.

```
756 {
757     return (*this);
758 }
```

8.455.3.64 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 }
```

8.455.3.65 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 }
```

8.455.3.66 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 }
```

8.455.3.67 HxString HxVec3Double::toString () const

Value as a string.

```
119                              {
120     return HxString("(") + makeString(_values[0]) + ", "
121             + makeString(_values[1]) + ", "
122             + makeString(_values[2]) + ")";
123 }
```

8.455.4 Friends And Related Function Documentation

8.455.4.1 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 }
```

8.455.4.2 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 }
```

8.455.4.3 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 }
```

8.455.4.4 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 }
```

8.455.5 Member Data Documentation

8.455.5.1 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.

8.455.5.2 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:

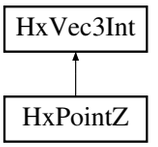
- **HxVec3Double.h**
- HxVec3Double.c

8.456 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. 1164).

- **operator HxScalarDouble** () const
  Cast to **HxScalarDouble** (p. 1145).
- **operator HxVec2Int** () const
  Cast to **HxVec2Int** (p. 1281).
- **operator HxVec2Double** () const
  Cast to **HxVec2Double** (p. 1262).
- **operator HxVec3Double** () const
  Cast to **HxVec3Double** (p. 1301).
- **operator HxComplex** () const
  Cast to **HxComplex** (p. 506).

Operators

Mathematical definition: **Binary operations on pixel values** (p. 5)

- 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 on pixel values** (p. 4)

- 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 on pixel values** (p. 5)

- 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)

8.456.1 Detailed Description

Class definition vector of 3 integers.

8.456.2 Constructor & Destructor Documentation

8.456.2.1 HxVec3Int::HxVec3Int () [inline]

Default constructor.

```
322 {
323 }
```

8.456.2.2 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 }
```

8.456.2.3 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 }
```

8.456.3 Member Function Documentation

8.456.3.1 int HxVec3Int::dim () const [inline]

Dimensionality.

```
349 {
350     return 3;
351 }
```

8.456.3.2 int HxVec3Int::x () const [inline]

Value of first element.

```
355 {
356     return _values[0];
357 }
```

8.456.3.3 int HxVec3Int::y () const [inline]

Value of second element.

```
361 {
362     return _values[1];
363 }
```

8.456.3.4 int HxVec3Int::z () const [inline]

Value of third element.

```
367 {
368     return _values[2];
369 }
```

8.456.3.5 int HxVec3Int::getValue (int *dimension*) const [inline]

Element in given dimension.

```
373 {
374     return _values[dimension - 1];
375 }
```

8.456.3.6 HxVec3Int::operator HxScalarInt () const

Cast to **HxScalarInt** (p. 1164).

```
28 {
29     return _values[0];
30 }
```

8.456.3.7 HxVec3Int::operator HxScalarDouble () const

Cast to **HxScalarDouble** (p. 1145).

```
33 {
34     return (double) _values[0];
35 }
```

8.456.3.8 HxVec3Int::operator HxVec2Int () const

Cast to **HxVec2Int** (p. 1281).

```
39 {
40     return HxVec2Int(_values[0], _values[1]);
41 }
```

8.456.3.9 HxVec3Int::operator HxVec2Double () const

Cast to **HxVec2Double** (p. 1262).

```
44 {
45     return HxVec2Double(_values[0], _values[1]);
46 }
```

8.456.3.10 HxVec3Int::operator HxVec3Double () const

Cast to **HxVec3Double** (p. 1301).

```
49 {
50     return HxVec3Double(_values[0], _values[1], _values[2]);
51 }
```

8.456.3.11 HxVec3Int::operator HxComplex () const

Cast to **HxComplex** (p. 506).

```
54 {
55     return HxComplex(_values[0], _values[1]);
56 }
```

8.456.3.12 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 }
```

8.456.3.13 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 }
```

8.456.3.14 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 }
```

8.456.3.15 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 }
```

8.456.3.16 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 }
```

8.456.3.17 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 }
```

8.456.3.18 HxVec3Int HxVec3Int::operator- () const [inline]

Negation.

```
427 {
428     return HxVec3Int(-_values[0], -_values[1], -_values[2]);
429 }
```

8.456.3.19 HxVec3Int HxVec3Int::complement () const [inline]

Complement.

```
433 {
434     return HxVec3Int(~_values[0], ~_values[1], ~_values[2]);
435 }
```

8.456.3.20 HxVec3Int HxVec3Int::abs () const [inline]

Absolute value.

```
439 {
440     return HxVec3Int(::abs(_values[0]), ::abs(_values[1]), ::abs(_values[2]));
441 }
```

8.456.3.21 HxVec3Int HxVec3Int::ceil () const [inline]

Ceiling.

```
445 {
446     return *this;
447 }
```

8.456.3.22 HxVec3Int HxVec3Int::floor () const [inline]

Floor.

```
451 {
452     return *this;
453 }
```

8.456.3.23 HxVec3Int HxVec3Int::round () const [inline]

Round.

```
457 {
458     return *this;
459 }
```

8.456.3.24 HxScalarInt HxVec3Int::sum () const [inline]

Sum.

```
463 {
464     return _values[0] + _values[1] + _values[2];
465 }
```

8.456.3.25 HxScalarInt HxVec3Int::product () const [inline]

Product.

```
469 {
470     return _values[0] * _values[1] * _values[2];
471 }
```

8.456.3.26 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 }
```

8.456.3.27 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 }
```

8.456.3.28 HxScalarInt HxVec3Int::norm1 () const

L1 norm.

```
76 {
77     return ::abs(_values[0]) + ::abs(_values[1]) + ::abs(_values[2]);
78 }
```

8.456.3.29 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 }
```

8.456.3.30 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 }
```

8.456.3.31 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 }
```

8.456.3.32 HxVec3Double HxVec3Int::sin () const

Sine.

```
108 {
109     return HxVec3Double(::sin(double(_values[0])),
110                         ::sin(double(_values[1])),
111                         ::sin(double(_values[2])));
112 }
```

8.456.3.33 HxVec3Double HxVec3Int::cos () const

Cosine.

```
116 {
117     return HxVec3Double(::cos(double(_values[0])),
118                         ::cos(double(_values[1])),
119                         ::cos(double(_values[2])));
120 }
```

8.456.3.34 HxVec3Double HxVec3Int::tan () const

Tangent.

```
124 {
125     return HxVec3Double(::tan(double(_values[0])),
126                         ::tan(double(_values[1])),
127                         ::tan(double(_values[2])));
128 }
```

8.456.3.35 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 }
```

8.456.3.36 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 }
```

8.456.3.37 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 }
```

8.456.3.38 HxScalarDouble HxVec3Int::atan2 () const

Arc tangent.

```
148 {
149     return HxScalarDouble(::atan2(double(_values[0]), double(_values[1])));
150 }
```

8.456.3.39 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 }
```

8.456.3.40 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 }
```

8.456.3.41 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 }
```

8.456.3.42 HxVec3Double HxVec3Int::exp () const

Exponent.

```
186 {
187     return HxVec3Double(::exp(double(_values[0])),
188                         ::exp(double(_values[1])),
189                         ::exp(double(_values[2])));
190 }
```

8.456.3.43 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 }
```

8.456.3.44 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 }
```

8.456.3.45 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 }
```

8.456.3.46 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 }
```

8.456.3.47 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 }
```

8.456.3.48 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 }
```

8.456.3.49 HxVec3Int HxVec3Int::min (const HxVec3Int & *v*) const [inline]

Minimum.

```
543 {
544     return (operator<(v)) ? (*this) : v;
545 }
```

8.456.3.50 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 }
```

8.456.3.51 HxVec3Int HxVec3Int::max (const HxVec3Int & *v*) const [inline]

Maximum.

```
558 {
559     return (operator>(v)) ? (*this) : v;
560 }
```

8.456.3.52 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 }
```

8.456.3.53 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 }
```

8.456.3.54 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 }
```

8.456.3.55 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 }
```

8.456.3.56 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 }
```

8.456.3.57 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 }
```

8.456.3.58 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 }
```

8.456.3.59 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 }
```

8.456.3.60 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 }
```

8.456.3.61 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 }
```

8.456.3.62 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 }
```

8.456.3.63 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 }
```

8.456.3.64 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 }
```

8.456.3.65 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 }
```

8.456.3.66 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 }
```

8.456.3.67 HxString HxVec3Int::toString () const

Value as a string.

```
224                            {
225     return HxString("(") + makeString(_values[0]) + ", "
226             + makeString(_values[1]) + ", "
227             + makeString(_values[2]) + ")";
228 }
```

8.456.4 Friends And Related Function Documentation

8.456.4.1 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 }
```

8.456.4.2 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 }
```

8.456.4.3 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 }
```

8.456.4.4 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 }
```

8.456.5 Member Data Documentation

8.456.5.1 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.

8.456.5.2 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

8.457 HxVec3Tem Class Template Reference

Template class for representation of vector of 3 scalars of type T.

`#include <HxVec3Tem.h>`

Public Methods

- **HxVec3Tem** ()
- **HxVec3Tem** (T x, T y, T z)
- **HxVec3Tem** (const HxVec3Tem< T > &v)
- **HxVec3Tem** (int v)
- **HxVec3Tem** (double v)
- **HxVec3Tem** (const **HxVec3Int** &v)
- **HxVec3Tem** (const **HxVec3Double** &v)
- void * **operator new** (size_t, void *=0)
- T **x** () const
- T **y** () const
- T **z** () const
- T & **value** (int dimension)
- **operator int** () const
- **operator double** () const
- **operator HxVec3Int** () const
- **operator HxVec3Double** () const
- HxVec3Tem< T > & **operator+=** (const HxVec3Tem< T > &v)
- HxVec3Tem< T > & **operator-=** (const HxVec3Tem< T > &v)
- HxVec3Tem< T > & **operator** *= (const HxVec3Tem< T > &v)
- HxVec3Tem< T > & **operator/=** (const HxVec3Tem< T > &v)
- STD_OSTREAM & **put** (STD_OSTREAM &os) const

Friends

- HxVec3Tem< T > **operator+** (const HxVec3Tem< T > &v1, const HxVec3Tem< T > &v2)
- HxVec3Tem< T > **operator-** (const HxVec3Tem< T > &v1, const HxVec3Tem< T > &v2)
- HxVec3Tem< T > **operator** * (const HxVec3Tem< T > &v1, const HxVec3Tem< T > &v2)
- HxVec3Tem< T > **operator/** (const HxVec3Tem< T > &v1, const HxVec3Tem< T > &v2)

8.457.1 Detailed Description

template<class T> class HxVec3Tem< T >

Template class for representation of vector of 3 scalars of type T.

The documentation for this class was generated from the following files:

- **HxVec3Tem.h**
- HxVec3Tem.c

8.458 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**

8.458.1 Detailed Description

Class definition for vectors.

The vector may have arbitrary size.

8.458.2 Constructor & Destructor Documentation

8.458.2.1 HxVector::HxVector () [inline]

Empty vector.

```
216 {
217     _n = 0;
218     _data = 0;
219 }
```

8.458.2.2 HxVector::HxVector (int *n*) [inline]

Empty vector of given size.

```
222 {
223     _n = n;
224     _data = new double[_n];
225 }
```

8.458.2.3 HxVector::HxVector (int *n*, double * *data*) [inline]

vector with given data.

```
228 {
229     _n = n;
230     _data = data;
231 }
```

8.458.2.4 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 }
```

8.458.2.5 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 }
```

8.458.2.6 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 }
```

8.458.2.7 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 }
```

8.458.2.8 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 }
```

8.458.3 Member Function Documentation

8.458.3.1 int HxVector::nElem () const [inline]

Number of elements.

```
279 {
280     return _n;
281 }
```

8.458.3.2 int HxVector::valid () const [inline]

Indicates whether the vector is valid.

```
285 {
286     return (_n != 0);
287 }
```

8.458.3.3 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 }
```

8.458.3.4 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 }
```

8.458.3.5 double & HxVector::operator[ ] (int *i*) const [inline]

Subscripting, start with 0.

```
317 {
318     return _data[i];
319 }
```

8.458.3.6 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 }
```

8.458.3.7 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 }
```

8.458.3.8 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 }
```

8.458.3.9 HxVector HxVector::add (const HxVector & *b*) const

Addition.

Equivalent to : a+b

```
124 {
125     return *this+b;
126 }
```

8.458.3.10 HxVector HxVector::add (const double *val*) const

Addition.

Equivalent to : a+val

```
130 {
131     return *this+val;
132 }
```

8.458.3.11 HxVector HxVector::sub (const HxVector & *b*) const

Subtraction.

Equivalent to : a-b

```
136 {
137     return *this-b;
138 }
```

8.458.3.12 HxVector HxVector::sub (const double *val*) const

Subtraction.

Equivalent to : a-val

```
142 {
143     return *this-val;
144 }
```

8.458.3.13 HxVector HxVector::mul (const HxVector & *b*) const

Multiplication.

Equivalent to : a∗b

```
148 {
149     return *this*b;
150 }
```

8.458.3.14 HxVector HxVector::mul (const HxMatrix & *m*) const

Multiplication.

Equivalent to : a∗v

```
160 {
161     return *this*m;
162 }
```

8.458.3.15 HxVector HxVector::mul (const double *val*) const

Multiplication.

Equivalent to : a∗val

```
154 {
155     return *this*val;
156 }
```

8.458.3.16 HxVector HxVector::div (const double *val*) const

Division.

Equivalent to : a/val

```
166 {
167     return *this/val;
168 }
```

8.458.3.17 HxVector HxVector::sin () const

Apply sin to each element.

```
172 {
173     return map(::sin);
174 }
```

8.458.3.18 HxVector HxVector::cos () const

Apply cos to each element.

```
178 {
179     return map(::cos);
180 }
```

8.458.3.19 HxVector HxVector::tan () const

Apply tan to each element.

```
184 {
185     return map(::tan);
186 }
```

8.458.3.20 HxVector HxVector::sinh () const

Apply sinh to each element.

```
190 {
191     return map(::sinh);
192 }
```

8.458.3.21 HxVector HxVector::cosh () const

Apply cosh to each element.

```
196 {
197     return map(::cosh);
198 }
```

8.458.3.22 HxVector HxVector::tanh () const

Apply tanh to each element.

```
202 {
203     return map(::tanh);
204 }
```

8.458.3.23 HxVector HxVector::exp () const

Apply exp to each element.

```
208 {
209     return map(::exp);
210 }
```

8.458.3.24 HxVector HxVector::log () const

Apply log to each element.

```
214 {
215     return map(::log);
216 }
```

8.458.3.25 HxVector HxVector::sqrt () const

Apply sqrt to each element.

```
220 {
221     return map(::sqrt);
222 }
```

8.458.3.26 HxVector HxVector::abs () const

Apply abs to each element.

```
226 {
227     return map(::fabs);
228 }
```

8.458.3.27 HxVector HxVector::sgn () const

Apply sgn to each element.

```
234 {
235     return map(::sgn);
236 }
```

8.458.3.28 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 }
```

8.458.4 Friends And Related Function Documentation

8.458.4.1 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 }
```

8.458.4.2 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 }
```

8.458.4.3 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 }
```

8.458.4.4 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 }
```

8.458.4.5 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 }
```

8.458.4.6 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 }
```

8.458.4.7 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 }
```

8.458.4.8 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 }
```

8.458.4.9 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 }
```

8.458.4.10 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 }
```

8.458.4.11 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 }
```

8.458.4.12 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 }
```

8.458.4.13 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

8.459 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**

8.459.1 Detailed Description

Class definition for vectors in R2 (real-value coordinates).

8.459.2 Constructor & Destructor Documentation

8.459.2.1 HxVectorR2::HxVectorR2 () [inline]

Constructor.

```
87                          : _data(0,0)
88 {
89 }
```

8.459.2.2 HxVectorR2::HxVectorR2 (double *d1*, double *d2*) [inline]

Constructor.

```
92                                              : _data(d1, d2)
93 {
94 }
```

8.459.2.3 HxVectorR2::HxVectorR2 (const HxPointR2 & *p1*, const HxPointR2 & *p2*)

Constructor : p1-p2.

```
16 {
17     _data = HxVec2Double(p1.x()-p2.x(), p1.y()-p2.y());
18 }
```

8.459.3 Member Function Documentation

8.459.3.1 double HxVectorR2::x () const [inline]

Get the first element of the vector.

```
98 {
99     return _data.x();
100 }
```

8.459.3.2 double HxVectorR2::y () const [inline]

Get the second element of the vector.

```
104 {
105     return _data.y();
106 }
```

8.459.3.3 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 }
```

8.459.3.4 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 }
```

8.459.3.5 HxVectorR2 HxVectorR2::mul (double *arg*) const [inline]

Multiply this vector with a scalar.

```
122 {
123     return HxVectorR2( _data.x()*arg, _data.y()*arg);
124 }
```

8.459.3.6 HxVectorR2 HxVectorR2::div (double *arg*) const [inline]

Divide this vector by a scalar.

```
128 {
129     return HxVectorR2( _data.x()/arg, _data.y()/arg);
130 }
```

8.459.3.7 double HxVectorR2::dot (const HxVectorR2 & *arg*) const [inline]

Dot product.

```
134 {
135     return _data.dot(arg._data).x();
136 }
```

8.459.3.8 double HxVectorR2::cross2D (const HxVectorR2 & *arg*) const [inline]

Cross product (?).

```
140 {
141     return _data.x()*arg.y() + _data.y()*arg.x();
142 }
```

8.459.3.9 double HxVectorR2::magnitude () const [inline]

Magnitude.

```
146 {
147     return _data.norm2().x();
148 }
```

8.459.3.10 double HxVectorR2::squaredMagnitude () const [inline]

Squared magnitude.

```
152 {
153     return _data.x()*_data.x() + _data.y()*_data.y();
154 }
```

8.459.3.11 HxVectorR2 HxVectorR2::normal () const [inline]

Normal.

```
158 {
159     return HxVectorR2(_data.y(), -_data.x());
160 }
```

8.459.3.12 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

8.460 QThinning Class Template Reference

This should be implemented (Rein) as repetitive substraction from the image of result of HitOrMiss.

Public Types

- typedef QThinning::structPointValue **PointValueT**
- typedef Qset< PointValueT > **QT**
- typedef PointValueT **Neighbors**
- typedef FcvArray< Neighbors > **VecNeighbors**
- enum **ActionT** { **queueAction**, **removeAction**, **writeAction**, **stopAction** }

Public Methods

- **QThinning** (**HxTagList** &tags, unsigned x, unsigned y)
- ~**QThinning** ()
- bool **globalPixelInit** (PointValueT &, ArithT &arith, ImgT img, MaskT)
- bool **wantFreshStartLocalPixelInit** ()

- bool **localPixelInit** (PointValueT &vp, ArithT &arith, ImgT img, MaskT mask, bool &continueloop)
- void **first** (const PointValueT &val, ArithT arith, ImgT img, MaskT mask)
- void **calculate** (VecNeighbors &vecneighbors)
- ActionT **result3** ()
- void **getItemToQueue** (PointValueT &vp)
- void **getItemToRemove** (PointValueT &, PointValueT &)
- bool **killThisOne** (const PointValueT &)
- void **getItemToWrite** (PointT &point, ArithT &arith)
- bool **wantAnotherLoop** ()

Static Public Methods

- **HxString className** ()

8.460.1 Detailed Description

template<class ArithT, class ImgT, class MaskT> class QThinning< ArithT, ImgT, MaskT >

This should be implemented (Rein) as repetitive substraction from the image of result of HitOrMiss.

As they are not rotation invariant, the structuring elements of have to be rotated

This is defined only for binary images 1 - pixels = object 0 - pixels = background

The documentation for this class was generated from the following file:

- HxThinning.c

8.461 RGB2Intensity Class Template Reference

Pixel functor for computation of RGB2Intensity.

Public Types

- typedef **HxTagTransInVar TransVarianceCategory**
  Functor is translation invariant.

Public Methods

- **RGB2Intensity** (**HxTagList** &)
  Constructor : get parameters from taglist.
- DstValT **doIt** (const SrcValT &x)
  Actual operation.

Static Public Methods

- **HxString className** ()

  The name : "RGB2Intensity".

8.461.1 Detailed Description

template<class DstValT, class SrcValT> class RGB2Intensity< DstValT, SrcValT >

Pixel functor for computation of RGB2Intensity.

8.461.2 Member Typedef Documentation

8.461.2.1 template<class DstValT, class SrcValT> typedef HxTagTransInVar RGB2Intensity::TransVarianceCategory

Functor is translation invariant.

8.461.3 Constructor & Destructor Documentation

8.461.3.1 template<class DstValT, class SrcValT> RGB2Intensity< DstValT, SrcValT >::RGB2Intensity (HxTagList & *tl*)

Constructor : get parameters from taglist.

```
38 {
39 }
```

8.461.4 Member Function Documentation

8.461.4.1 template<class DstValT, class SrcValT> DstValT RGB2Intensity< DstValT, SrcValT >::doIt (const SrcValT & *x*) [inline]

Actual operation.

```
44 {
45     return 0.212671*x.x() + 0.715160*x.y() + 0.072169*x.z();
46 }
```

8.461.4.2 template<class DstValT, class SrcValT> HxString RGB2Intensity< DstValT, SrcValT >::className () [static]

The name : "RGB2Intensity".

```
51 {
52     return HxString("RGB2Intensity");
53 }
```

The documentation for this class was generated from the following file:

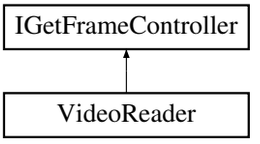
- HxRGB2Intensity.c

8.462 VideoReader Class Reference

Read frames from a video file.

`#include <VideoReader.h>`

Inheritance diagram for VideoReader::



Public Methods

- **VideoReader** (const char *name, int cacheSize=20)
- virtual ~**VideoReader** ()
- int **isNull** ()
- int **getLength** ()
- BYTE * **getFrame** (int n)
- int **getFrameWidth** ()
- int **getFrameHeight** ()
- STDMETHODIMP **NextData** (BYTE *data)
- STDMETHODIMP **SetDataSizes** (int width, int height, REFERENCE_TIME timePerFrame)
- STDMETHODIMP **QueryInterface** (const IID &riid, void **ppv)
- **STDMETHODIMP_** (ULONG) AddRef()
- **STDMETHODIMP_** (ULONG) Release()

8.462.1 Detailed Description

Read frames from a video file.

The documentation for this class was generated from the following files:

- **VideoReader.h**
- VideoReader.c

8.463 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**

8.463.1 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

_dimSizes
HxImageTem, 695
_imageDimensionality
HxImageSignature, 688
_pixelDimensionality
HxImageSignature, 688
_pixelPrecision
HxImageSignature, 688
_pixelType
HxImageSignature, 688
~DeviceEnumerator
DeviceEnumerator, 399
~HxArrowR2
HxArrowR2, 401
~HxBSplineBasis
HxBSplineBasis, 473
~HxBSplineCurve
HxBSplineCurve, 480
~HxBSplineInterval
HxBSplineInterval, 492
~HxBlob2d
HxBlob2d, 403
~HxBlob2dFeature
HxBlob2dFeature, 407
~HxBlob2dFeatureTem
HxBlob2dFeatureTem, 409
~HxBlob2dRelation
HxBlob2dRelation, 412
~HxExportExtraIdentMaskCentralMoments
HxExportExtraIdentMaskCentralMoments, 532
~HxExportExtraIdentMaskMean
HxExportExtraIdentMaskMean, 535
~HxExportExtraIdentMaskMedian
HxExportExtraIdentMaskMedian, 537
~HxExportExtraIdentMaskMoments
HxExportExtraIdentMaskMoments, 540
~HxExportExtraIdentMaskStdev
HxExportExtraIdentMaskStdev, 542
~HxExportExtraIdentMaskSum
HxExportExtraIdentMaskSum, 545
~HxExportExtraWeightMaskSum
HxExportExtraWeightMaskSum, 547
~HxHistogram
HxHistogram, 555
~HxImageData
HxImageData, 587
~HxImageList
HxImageList, 618
~HxImageRep
HxImageRep, 626
~HxImageSeq
HxImageSeq, 649
~HxImageSeqDXMedia
HxImageSeqDXMedia, 659
~HxImageSeqData
HxImageSeqData, 655
~HxImageSeqIter
HxImageSeqIter, 663
~HxImageSeqMDC
HxImageSeqMDC, 667
~HxImageTem
HxImageTem, 694
~HxImageTem2d
HxImageTem2d, 697
~HxImageTem3d
HxImageTem3d, 699
~HxImgFtorBpo
HxImgFtorBpo, 702
~HxImgFtorDescription
HxImgFtorDescription, 704
~HxImgFtorDiy
HxImgFtorDiy, 706
~HxImgFtorExportExtra
HxImgFtorExportExtra, 709
~HxImgFtorGenConv2d
HxImgFtorGenConv2d, 713
~HxImgFtorGenConv2dK1d
HxImgFtorGenConv2dK1d, 716
~HxImgFtorGenConv2dSep
HxImgFtorGenConv2dSep, 719
~HxImgFtorGenConv3d
HxImgFtorGenConv3d, 722
~HxImgFtorGenConv3dK1d
HxImgFtorGenConv3dK1d, 725
~HxImgFtorI1
HxImgFtorI1, 730
~HxImgFtorI1Cast
HxImgFtorI1Cast, 734
~HxImgFtorI2

HxImgFtorI2, 742
~HxImgFtorI2Cast
HxImgFtorI2Cast, 749
~HxImgFtorI3
HxImgFtorI3, 759
~HxImgFtorI3Cast
HxImgFtorI3Cast, 763
~HxImgFtorI4
HxImgFtorI4, 771
~HxImgFtorI4Cast
HxImgFtorI4Cast, 774
~HxImgFtorIM
HxImgFtorIM, 779
~HxImgFtorIMCast
HxImgFtorIMCast, 781
~HxImgFtorIMN
HxImgFtorIMN, 786
~HxImgFtorIMNCast
HxImgFtorIMNCast, 788
~HxImgFtorInOut
HxImgFtorInOut, 793
~HxImgFtorKernelNgb2d
HxImgFtorKernelNgb2d, 796
~HxImgFtorKeyNameTable
HxImgFtorKeyNameTable, 803
~HxImgFtorMNpo
HxImgFtorMNpo, 805
~HxImgFtorMpo
HxImgFtorMpo, 809
~HxImgFtorNgb2d
HxImgFtorNgb2d, 812
~HxImgFtorNgb2dExtra
HxImgFtorNgb2dExtra, 815
~HxImgFtorNgb2dExtra2
HxImgFtorNgb2dExtra2, 818
~HxImgFtorQueueBased
HxImgFtorQueueBased, 823
~HxImgFtorRecGenConv2d
HxImgFtorRecGenConv2d, 829
~HxImgFtorRecGenConv2dK1d
HxImgFtorRecGenConv2dK1d, 832
~HxImgFtorRgb2d
HxImgFtorRgb2d, 836
~HxImgFtorRgb3d
HxImgFtorRgb3d, 839
~HxImgFtorRuleBase
HxImgFtorRuleBase, 842
~HxImgFtorSet
HxImgFtorSet, 848
~HxImgFtorSetBorder2d
HxImgFtorSetBorder2d, 851
~HxImgFtorSetBorder3d
HxImgFtorSetBorder3d, 853
~HxImgFtorTable
HxImgFtorTable, 857
~HxImgFtorUpo
HxImgFtorUpo, 860
~HxImgFunctor
HxImgFunctor, 863
~HxKerNgbNormCorrelation
HxKerNgbNormCorrelation, 974
~HxKernel1d
HxKernel1d, 971
~HxKernel2d
HxKernel2d, 972
~HxKernel3d
HxKernel3d, 972
~HxLocalInterpol
HxLocalInterpol, 978
~HxMatrix
HxMatrix, 984
~HxMfBpo
HxMfBpo, 1008
~HxMfDiy
HxMfDiy, 1010
~HxMfExportExtra
HxMfExportExtra, 1012
~HxMfGenConv
HxMfGenConv, 1015
~HxMfIdentity
HxMfIdentity, 1017
~HxMfKernelNgb
HxMfKernelNgb, 1019
~HxMfMNpo
HxMfMNpo, 1022
~HxMfMpo
HxMfMpo, 1025
~HxMfNgb
HxMfNgb, 1028
~HxMfQueueBased
HxMfQueueBased, 1030
~HxMfResize
HxMfResize, 1032
~HxMfUpo
HxMfUpo, 1034
~HxNJet
HxNJet, 1067
~HxNameTable
HxNameTable, 1036
~HxNgbBernsen
HxNgbBernsen, 1039
~HxNgbDefuz
HxNgbDefuz, 1040
~HxNgbHilditch
HxNgbHilditch, 1042
~HxNgbIsMaxGradDir2d
HxNgbIsMaxGradDir2d, 1045
~HxNgbKuwahara

HxNgbKuwahara, 1048
~HxNgbLWshed2d
HxNgbLWshed2d, 1054
~HxNgbNonMaxSuppression2d
HxNgbNonMaxSuppression2d, 1058
~HxNgbPercentile2d
HxNgbPercentile2d, 1062
~HxPixelAllocator
HxPixelAllocator, 1072
~HxPolyline2d
HxPolyline2d, 1081
~HxRcObject
HxRcObject, 1083
~HxRcPtr
HxRcPtr, 1083
~HxRegData
HxRegData, 1086
~HxRegKey
HxRegKey, 1099
~HxRegValue
HxRegValue, 1105
~HxRegistry
HxRegistry, 1090
~HxSF
HxSF, 1187
~HxSampledBSplineCurve
HxSampledBSplineCurve, 1133
~HxSampledBSplineInterval
HxSampledBSplineInterval, 1143
~HxSegmentation2d
HxSegmentation2d, 1183
~HxTag
HxTag, 1193
~HxTagList
HxTagList, 1197
~HxTagTem
HxTagTem, 1203
~HxVector
HxVector, 1345
~QThinning
QThinning, 1360
~VideoReader
VideoReader, 1363

abs
HxComplex, 515
HxMatrix, 1000
HxScalarDouble, 1154
HxScalarInt, 1172
HxVec2Double, 1271
HxVec2Int, 1290
HxVec3Double, 1310
HxVec3Int, 1330
HxVector, 1352

acos
HxComplex, 518
HxScalarDouble, 1156
HxScalarInt, 1175
HxVec2Double, 1274
HxVec2Int, 1292
HxVec3Double, 1313
HxVec3Int, 1333
add
HxBlob2dRelation, 412
HxMatrix, 997
HxPointR2, 1076
HxVector, 1349
HxVectorR2, 1358
addArgument
HxImgFtorDescription, 704
HxImgFtorKey, 801
addBlob
HxSegmentation2d, 1184, 1185
addFeature
HxBlob2d, 404
addImgFtorObserver
HxImgFtorTable, 858
addRef
HxRcObject, 1083
addRelation
HxSegmentation2d, 1185
address
HxPixelAllocator, 1072
addTag
HxTagList, 1198
AllC
HxSampledBSplineCurve, 1138
allKnots
HxBSplineBasis, 474
HxLocalInterpol, 978
allocate
HxPixelAllocator, 1072
Allocator
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680

HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
allP
HxBSplineCurve, 482
HxLocalInterpol, 978
HxSampledBSplineCurve, 1140
allSampledT
HxSampledBSplineCurve, 1134
and
HxComplex, 523
HxScalarDouble, 1161
HxScalarInt, 1179
HxVec2Double, 1278
HxVec2Int, 1297
HxVec3Double, 1318
HxVec3Int, 1337
arg
HxComplex, 515
argument
HxMfResize, 1033
ArithImageSigType
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
HxImageTem, 694
ArithImageSigTypeDouble
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
HxImageTem, 694
ArithType
HxBpoAddAssign, 419
HxBpoInfAssign, 437
HxBpoMaxAssign, 445
HxBpoMinAssign, 448
HxBpoMulAssign, 452
HxBpoSubAssign, 463
HxBpoSupAssign, 467
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
HxImageTem, 694
HxKernel1d, 971
HxKernel2d, 972
HxKernel3d, 972
ArithTypeDouble
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672

HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
HxImageTem, 694
HxRgbBinary, 1108
HxRgbCMY, 1110
HxRgbDirect, 1112
HxRgbDirectNC, 1113
HxRgbHSI, 1115
HxRgbLab, 1117
HxRgbLabel, 1119
HxRgbLogMag, 1121
HxRgbLuv, 1122
HxRgbOOO, 1124
HxRgbStretch, 1126
HxRgbXYZ, 1128

asin
HxComplex, 518
HxScalarDouble, 1156
HxScalarInt, 1175
HxVec2Double, 1274
HxVec2Int, 1292
HxVec3Double, 1313
HxVec3Int, 1333

assign
HxRcObject, 1083

atan
HxComplex, 518
HxScalarDouble, 1157
HxScalarInt, 1175
HxVec2Double, 1274
HxVec2Int, 1293
HxVec3Double, 1313
HxVec3Int, 1333

atan2
HxComplex, 519
HxScalarDouble, 1157
HxScalarInt, 1175
HxVec2Double, 1274
HxVec2Int, 1293
HxVec3Double, 1313
HxVec3Int, 1333

atof
HxStringNative.h, 358

atoi
HxStringNative.h, 358

atol
HxStringNative.h, 358

AVI_F
HxImageSeq, 648

B
HxBSplineBasis, 474
HxBSplineCurve, 483
HxSampledBSplineCurve, 1136

back_insert_iterator
HxBlob2dList, 410
HxPointList, 1074
HxRegKeyList, 1104
HxRegValueList, 1107
HxStringList, 1192
HxValueList, 1262

BAll
HxSampledBSplineCurve, 1137

BasicFeatures
HxGetBlobFeatures.h, 231

BasicFeaturesList
HxGetBlobFeatures.h, 231

basis
HxBSplineCurve, 481

begin
HxBoundingBox, 415
HxBSplineInterval, 492
HxImageList, 617
HxImageSeq, 652
HxNgbIsMaxGradDir2d, 1046
HxNgbNonMaxSuppression2d, 1058
HxSampledBSplineInterval, 1144

binaryPixOp
HxImageData, 593
HxImageList, 619
HxImageRep, 629, 630

binToValue
HxHistogram, 558

binWidth
HxHistogram, 557

broadest
HxImageSignature, 689

C
HxBSplineCurve, 484
HxSampledBSplineCurve, 1137

calculate
QThinning, 1361

callIt
HxImgFtorI1, 731
HxImgFtorI1Cast, 734
HxImgFtorI2, 743
HxImgFtorI2Cast, 749
HxImgFtorI3, 759
HxImgFtorI3Cast, 764
HxImgFtorI4, 771
HxImgFtorI4Cast, 775
HxImgFtorIM, 779
HxImgFtorIMCast, 782
HxImgFtorIMN, 786
HxImgFtorIMNCast, 789
camera
HxMatrix, 993
ceil
HxComplex, 515
HxScalarDouble, 1154
HxScalarInt, 1172
HxVec2Double, 1271
HxVec2Int, 1290
HxVec3Double, 1310
HxVec3Int, 1330
center
HxBSplineCurve, 486
changeAllP
HxBSplineCurve, 487
HxSampledBSplineCurve, 1141
check
HxNgbLocalMode, 1050
checkBorderSize
HxImageData, 587
checkEqualImageSig
HxImageData, 587
checkEqualImageSigAndSizes
HxImageData, 587
checkEqualImageSizes
HxImageData, 587
checkEqualImageSizesDim
HxImageData, 587
checkImageDimension
HxImageData, 587
checkLargerImageSigAndSizes
HxImageData, 587
checkPixelDimension
HxImageData, 587
checkProperKernelSigAndSizes
HxImageData, 587
chiSquare
HxHistogram, 570
chiSquareNorm
HxHistogram, 570
ClassName
HxSizes.h, 352
HxStringNative.h, 358
className
HxBpoAdd, 419
HxBpoAddAssign, 420
HxBpoAddSat, 422
HxBpoAnd, 423
HxBpoBind2Val, 425
HxBpoCross, 426
HxBpoDiv, 428
HxBpoDot, 429
HxBpoEqual, 431
HxBpoGreaterEqual, 432
HxBpoGreaterThan, 434
HxBpoHighlightRegion, 435
HxBpoInf, 437
HxBpoInfAssign, 438
HxBpoLeftShift, 440
HxBpoLessEqual, 441
HxBpoLessThan, 443
HxBpoMax, 444
HxBpoMaxAssign, 446
HxBpoMin, 447
HxBpoMinAssign, 449
HxBpoMod, 450
HxBpoMul, 452
HxBpoMulAssign, 453
HxBpoNotEqual, 455
HxBpoOr, 456
HxBpoPow, 458
HxBpoRightShift, 459
HxBpoSqrDst, 461
HxBpoSub, 462
HxBpoSubAssign, 464
HxBpoSubSat, 465
HxBpoSup, 467
HxBpoSupAssign, 468
HxBpoXor, 470
HxDiyTranspose, 530
HxExportExtraIdentMaskCentralMoments, 534
HxExportExtraIdentMaskMean, 536
HxExportExtraIdentMaskMedian, 538
HxExportExtraIdentMaskMoments, 541
HxExportExtraIdentMaskStdev, 543
HxExportExtraIdentMaskSum, 545
HxExportExtraWeightMaskSum, 547
HxImgFtorQueueBased, 823
HxKernel1d, 971
HxKernel2d, 972
HxKernel3d, 972
HxKerNgbNormCorrelation, 976
HxNgbBernsen, 1040
HxNgbDefuz, 1041
HxNgbHilditch, 1043

HxNgbIsMaxGradDir2d, 1047
HxNgbKuwahara, 1049
HxNgbLocalMode, 1052
HxNgbLWshed2d, 1056
HxNgbNonMaxSuppression2d, 1059
HxNgbPercentile2d, 1063
HxRgbBinary, 1109
HxRgbCMY, 1110
HxRgbDirect, 1112
HxRgbDirectNC, 1114
HxRgbHSI, 1116
HxRgbLab, 1117
HxRgbLabel, 1119
HxRgbLogMag, 1121
HxRgbLuv, 1123
HxRgbOOO, 1125
HxRgbStretch, 1127
HxRgbXYZ, 1129
HxUpoAbs, 1206
HxUpoAcos, 1208
HxUpoArg, 1209
HxUpoAsin, 1210
HxUpoAtan, 1212
HxUpoAtan2, 1213
HxUpoCeil, 1215
HxUpoColSpace, 1216
HxUpoComplement, 1218
HxUpoConjugate, 1219
HxUpoCos, 1221
HxUpoCosh, 1222
HxUpoExp, 1224
HxUpoFloor, 1225
HxUpoLog, 1227
HxUpoLog10, 1228
HxUpoMax, 1230
HxUpoMin, 1231
HxUpoNegate, 1233
HxUpoNorm1, 1234
HxUpoNorm2, 1236
HxUpoNorm2Sqr, 1237
HxUpoNormInf, 1239
HxUpoProduct, 1240
HxUpoRound, 1242
HxUpoSin, 1243
HxUpoSinh, 1245
HxUpoSqrt, 1246
HxUpoSum, 1248
HxUpoTan, 1249
HxUpoTanh, 1251
HxUpoTriStateThreshold, 1253
QThinning, 1361
RGB2Intensity, 1362

clone
HxBlob2dFeature, 408
HxBlob2dFeatureTem, 409
HxImageData, 586
HxImageSeqIter, 664
HxRcObject, 1083
HxTag, 1194
HxTagTem, 1203

closestSample
HxSampledBSplineCurve, 1140

CnumType
HxNgbIsMaxGradDir2d, 1045
HxNgbNonMaxSuppression2d, 1057

CO
HxAlternateSequentialFilter.h, 83

compare
HxImgFtorKey, 802

complement
HxComplex, 515
HxScalarDouble, 1154
HxScalarInt, 1172
HxVec2Double, 1271
HxVec2Int, 1290
HxVec3Double, 1310
HxVec3Int, 1330

computeEntropyThreshold
HxHistogram, 552

computeIsodataThreshold
HxHistogram, 552

conjugate
HxComplex, 515

const_address
HxPixelAllocator, 1072

const_iterator
HxImageList, 617

const_pointer
HxPixelAllocator, 1072

const_reference
HxPixelAllocator, 1072

contains
HxBSplineInterval, 493
HxSampledBSplineInterval, 1144

continuousCurve
HxSampledBSplineCurve, 1134

controlP
HxBSplineCurve, 482
HxSampledBSplineCurve, 1141

convert
HxColor, 500
HxHistogram, 571

correct
VxStructureEval, 1364

cos
HxComplex, 518
HxMatrix, 999
HxScalarDouble, 1156

HxScalarInt, 1174
HxVec2Double, 1273
HxVec2Int, 1292
HxVec3Double, 1312
HxVec3Int, 1332
HxVector, 1351
cosh
HxComplex, 519
HxMatrix, 999
HxScalarDouble, 1157
HxScalarInt, 1176
HxVec2Double, 1275
HxVec2Int, 1293
HxVec3Double, 1314
HxVec3Int, 1334
HxVector, 1351
countBins
HxHistogram, 576
CPoly
HxSampledBSplineCurve, 1138
createNeighborCoordinates
HxImgFtorQueueBased, 823
createRootKey
HxRegKey, 1099
cropBegin
HxBSplineInterval, 494
cropEnd
HxBSplineInterval, 494
cross
HxComplex, 524
HxScalarDouble, 1162
HxScalarInt, 1180
HxVec2Double, 1279
HxVec2Int, 1298
HxVec3Double, 1319
HxVec3Int, 1339
cross2D
HxVectorR2, 1359
curveType
HxBSplineBasis, 473
HxBSplineCurve, 481
HxSampledBSplineCurve, 1134

data
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529
DataPtr2dDouble
HxFuncSet.c, 222
DataPtr2dVec2Double
HxFuncSet.c, 222
dataPtrClone
HxImageTem, 695
HxImageTem2d, 697
HxImageTem3d, 699
DataPtrType
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
HxImageTem, 694
dataType
HxHistogram, 556
dB
HxBSplineBasis, 475
HxBSplineCurve, 483
HxSampledBSplineCurve, 1137
dBAll
HxSampledBSplineCurve, 1137
dC
HxBSplineCurve, 484
HxSampledBSplineCurve, 1138
dCAll
HxSampledBSplineCurve, 1138
deallocate
HxPixelAllocator, 1072
decX
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 527, 528
HxDataPtr3dScalarTem, 528
decXYZ
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529
decY
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 527, 528
HxDataPtr3dScalarTem, 528
decZ
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528

HxDataPtr3dScalarTem, 528, 529
degree
HxBSplineBasis, 473
HxBSplineCurve, 481
depth
HxImageTem3d, 699
DeviceEnumerator
~DeviceEnumerator, 399
DeviceEnumerator, 399
getFilter, 399
getNames, 399
DeviceEnumerator, 399
diag
HxVector, 1349
dilateSF
HxSF, 1188
dim
HxComplex, 512
HxScalarDouble, 1151
HxScalarInt, 1169
HxVec2Double, 1268
HxVec2Int, 1287
HxVec3Double, 1307
HxVec3Int, 1327
dimensionality
HxHistogram, 556
HxImageData, 588
HxImageRep, 628
HxImageTem, 695
dimensionSize
HxHistogram, 556
HxImageData, 588
HxImageRep, 628
HxImageTem, 695
dirty
HxImageRep, 625
displace
HxArrowR2, 401
div
HxMatrix, 998
HxVector, 1350
HxVectorR2, 1359
diyOp
HxImageData, 604
HxImageRep, 638
doGetUnshared
HxRcObject, 1083
doIt
HxBpoAdd, 419
HxBpoAddAssign, 420
HxBpoAddSat, 422
HxBpoAnd, 423
HxBpoBind2Val, 425
HxBpoCross, 426
HxBpoDiv, 428
HxBpoDot, 429
HxBpoEqual, 431
HxBpoGreaterEqual, 432
HxBpoGreaterThan, 434
HxBpoHighlightRegion, 435
HxBpoInf, 437
HxBpoInfAssign, 438
HxBpoLeftShift, 440
HxBpoLessEqual, 441
HxBpoLessThan, 443
HxBpoMax, 444
HxBpoMaxAssign, 446
HxBpoMin, 447
HxBpoMinAssign, 449
HxBpoMod, 450
HxBpoMul, 452
HxBpoMulAssign, 453
HxBpoNotEqual, 455
HxBpoOr, 456
HxBpoPow, 458
HxBpoRightShift, 459
HxBpoSqrDst, 461
HxBpoSub, 462
HxBpoSubAssign, 464
HxBpoSubSat, 465
HxBpoSup, 467
HxBpoSupAssign, 468
HxBpoXor, 470
HxDiyTranspose, 530
HxExportExtraIdentMaskCentralMoments, 533
HxExportExtraIdentMaskMean, 536
HxExportExtraIdentMaskMedian, 538
HxExportExtraIdentMaskMoments, 540
HxExportExtraIdentMaskStdev, 543
HxExportExtraIdentMaskSum, 545
HxExportExtraWeightMaskSum, 547
HxImgFtorBpo, 703
HxImgFtorDiy, 707
HxImgFtorExportExtra, 710
HxImgFtorGenConv2d, 713
HxImgFtorGenConv2dK1d, 716
HxImgFtorGenConv2dSep, 719
HxImgFtorGenConv3d, 723
HxImgFtorGenConv3dK1d, 725
HxImgFtorI1Cast, 734
HxImgFtorI2Cast, 750
HxImgFtorI3Cast, 764
HxImgFtorI4Cast, 775
HxImgFtorIMCast, 782
HxImgFtorIMNCast, 789
HxImgFtorInOut, 793
HxImgFtorKernelNgb2d, 797

HxImgFtorMNpo, 806
HxImgFtorMpo, 809
HxImgFtorNgb2d, 812
HxImgFtorNgb2dExtra, 815
HxImgFtorNgb2dExtra2, 818
HxImgFtorQueueBased, 824
HxImgFtorRecGenConv2d, 830
HxImgFtorRecGenConv2dK1d, 832
HxImgFtorRgb2d, 837
HxImgFtorRgb3d, 839
HxImgFtorSet, 848
HxImgFtorSetBorder2d, 851
HxImgFtorSetBorder3d, 853
HxImgFtorUpo, 861
HxRgbBinary, 1108
HxRgbCMY, 1110
HxRgbDirect, 1112
HxRgbDirectNC, 1114
HxRgbHSI, 1115
HxRgbLab, 1117
HxRgbLabel, 1119
HxRgbLogMag, 1121
HxRgbLuv, 1123
HxRgbOOO, 1125
HxRgbStretch, 1127
HxRgbXYZ, 1128
HxUpoAbs, 1206
HxUpoAcos, 1207
HxUpoArg, 1209
HxUpoAsin, 1210
HxUpoAtan, 1212
HxUpoAtan2, 1213
HxUpoCeil, 1215
HxUpoColSpace, 1216
HxUpoComplement, 1218
HxUpoConjugate, 1219
HxUpoCos, 1221
HxUpoCosh, 1222
HxUpoExp, 1224
HxUpoFloor, 1225
HxUpoLog, 1227
HxUpoLog10, 1228
HxUpoMax, 1230
HxUpoMin, 1231
HxUpoNegate, 1233
HxUpoNorm1, 1234
HxUpoNorm2, 1236
HxUpoNorm2Sqr, 1237
HxUpoNormInf, 1239
HxUpoProduct, 1240
HxUpoRound, 1242
HxUpoSin, 1243
HxUpoSinh, 1245
HxUpoSqrt, 1246
HxUpoSum, 1248
HxUpoTan, 1249
HxUpoTanh, 1251
HxUpoTriStateThreshold, 1252
RGB2Intensity, 1362
doItDouble
HxRgbBinary, 1109
HxRgbCMY, 1110
HxRgbDirect, 1112
HxRgbDirectNC, 1114
HxRgbHSI, 1116
HxRgbLab, 1117
HxRgbLabel, 1119
HxRgbLogMag, 1121
HxRgbLuv, 1123
HxRgbOOO, 1125
HxRgbStretch, 1127
HxRgbXYZ, 1129
done
HxExportExtraIdentMaskCentralMoments, 533
dot
HxComplex, 524
HxScalarDouble, 1162
HxScalarInt, 1180
HxVec2Double, 1279
HxVec2Int, 1298
HxVec3Double, 1318
HxVec3Int, 1338
HxVectorR2, 1359
DstDataPtrArray
HxImgFtorIMNCast, 788
DstDataPtrType
HxImgFtorIMCast, 781
HxImgFtorIMNCast, 788
dT
HxSampledBSplineCurve, 1135
dTurnAngleAtC
HxBSplineCurve, 485
HxSampledBSplineCurve, 1139
dTurnAngleAtCAll
HxSampledBSplineCurve, 1139
dummyVec2Byte
HxVec2Byte.h, 381
dummyVec2Float
HxVec2Float.h, 382
dummyVec2Short
HxVec2Short.h, 382
dummyVec3Byte
HxVec3Byte.h, 383
dummyVec3Float
HxVec3Float.h, 383
dummyVec3Short
HxVec3Short.h, 384

dump
HxBSplineBasis, 476
HxBSplineCurve, 490
HxLocalInterpol, 977
HxPointR2, 1075
HxSampledBSplineCurve, 1141
HxVectorR2, 1357

end
HxBoundingBox, 416
HxBSplineInterval, 492
HxImageList, 617
HxImageSeq, 652
HxNgbIsMaxGradDir2d, 1046
HxNgbNonMaxSuppression2d, 1058
HxSampledBSplineInterval, 1144
erase
HxTagList, 1198
eraseAll
HxPointList, 1074
HxPointZList, 1079
HxStringList, 1192
HxValueList, 1262
eraseKey
HxRegistry, 1092
HxRegKey, 1100
eraseValue
HxRegistry, 1093
HxRegKey, 1102
erodeSF
HxSF, 1186
exp
HxComplex, 520
HxMatrix, 1000
HxScalarDouble, 1157
HxScalarInt, 1176
HxVec2Double, 1275
HxVec2Int, 1293
HxVec3Double, 1314
HxVec3Int, 1334
HxVector, 1351
exportC
HxRegistry, 1092
exportExtra
HxImageData, 592
exportOp
HxImageData, 590, 591
HxImageRep, 631
exportOpExtra
HxImageRep, 631
exportText
HxRegistry, 1091
extend
HxBoundingBox, 416
HxImageData, 586
extra
HxMfExportExtra, 1012
HxMfNgb, 1028
extra2
HxMfNgb, 1028

f
HxInstantiatorAbs, 864
HxInstantiatorAcos, 865
HxInstantiatorAdd, 865
HxInstantiatorAddReduce, 866
HxInstantiatorAddSat, 867
HxInstantiatorAddV, 867
HxInstantiatorAnd, 868
HxInstantiatorAndV, 869
HxInstantiatorArg, 869
HxInstantiatorAsin, 870
HxInstantiatorAtan, 870
HxInstantiatorAtan2, 871
HxInstantiatorCeil, 872
HxInstantiatorColSpace, 872
HxInstantiatorComplement, 873
HxInstantiatorConjugate, 873
HxInstantiatorCos, 874
HxInstantiatorCosh, 875
HxInstantiatorCross, 875
HxInstantiatorCrossV, 876
HxInstantiatorDiv, 876
HxInstantiatorDivV, 877
HxInstantiatorDot, 878
HxInstantiatorDotV, 878
HxInstantiatorEqual, 879
HxInstantiatorEqualV, 880
HxInstantiatorExp, 880
HxInstantiatorExpPix, 881
HxInstantiatorFloor, 881
HxInstantiatorGpi, 882
HxInstantiatorGreaterEqual, 883
HxInstantiatorGreaterEqualV, 883
HxInstantiatorGreaterThan, 884
HxInstantiatorGreaterThanV, 885
HxInstantiatorHighlightRegion, 885
HxInstantiatorImpPix, 886
HxInstantiatorInf, 886
HxInstantiatorInfReduce, 887
HxInstantiatorInfV, 888
HxInstantiatorLeftShift, 888
HxInstantiatorLeftShiftV, 889
HxInstantiatorLessEqual, 890
HxInstantiatorLessEqualV, 890
HxInstantiatorLessThan, 891
HxInstantiatorLessThanV, 892
HxInstantiatorLog, 892

HxInstantiatorLog10, 893
HxInstantiatorMax, 893
HxInstantiatorMaxReduce, 894
HxInstantiatorMaxV, 895
HxInstantiatorMin, 895
HxInstantiatorMinReduce, 896
HxInstantiatorMinV, 897
HxInstantiatorMod, 897
HxInstantiatorModV, 898
HxInstantiatorMul, 899
HxInstantiatorMulReduce, 899
HxInstantiatorMulV, 900
HxInstantiatorNegate, 901
HxInstantiatorNorm1, 901
HxInstantiatorNorm2, 902
HxInstantiatorNorm2Sqr, 902
HxInstantiatorNormInf, 903
HxInstantiatorNotEqual, 904
HxInstantiatorNotEqualV, 904
HxInstantiatorOr, 905
HxInstantiatorOrV, 906
HxInstantiatorPow, 906
HxInstantiatorPowV, 907
HxInstantiatorProduct, 908
HxInstantiatorRGB2Intensity, 908
HxInstantiatorRightShift, 909
HxInstantiatorRightShiftV, 909
HxInstantiatorRound, 910
HxInstantiatorSetPartImg, 915
HxInstantiatorSetVal, 915
HxInstantiatorSin, 916
HxInstantiatorSinh, 916
HxInstantiatorSpi, 917
HxInstantiatorSqrDst, 917
HxInstantiatorSqrt, 918
HxInstantiatorSub, 919
HxInstantiatorSubSat, 919
HxInstantiatorSubV, 920
HxInstantiatorSum, 921
HxInstantiatorSup, 921
HxInstantiatorSupReduce, 922
HxInstantiatorSupV, 922
HxInstantiatorTan, 923
HxInstantiatorTanh, 924
HxInstantiatorTriStateThreshold, 924
HxInstantiatorUpoMax, 925
HxInstantiatorUpoMin, 925
HxInstantiatorUpoThreshold, 926
HxInstantiatorVec2, 927
HxInstantiatorVec3, 927
HxInstantiatorXor, 928
HxInstantiatorXorV, 928
HxInstDiyTranspose, 929
HxInstExportExtraIdentMaskCentral-Moments, 930
HxInstExportExtraIdentMaskMean, 930
HxInstExportExtraIdentMaskMedian, 931
HxInstExportExtraIdentMaskMoments, 932
HxInstExportExtraIdentMaskStdev, 933
HxInstExportExtraIdentMaskSum, 933
HxInstExportExtraWeightMaskSum, 934
HxInstExpPpm, 935
HxInstGenConv2dAddInf, 935
HxInstGenConv2dAddMax, 936
HxInstGenConv2dAddMin, 937
HxInstGenConv2dAddSup, 938
HxInstGenConv2dK1dAddInf, 938
HxInstGenConv2dK1dAddMax, 939
HxInstGenConv2dK1dAddMin, 940
HxInstGenConv2dK1dAddSup, 941
HxInstGenConv2dK1dMulAdd, 941
HxInstGenConv2dMulAdd, 942
HxInstGenConv2dSepAddInf, 943
HxInstGenConv2dSepAddMax, 944
HxInstGenConv2dSepAddMin, 944
HxInstGenConv2dSepAddSup, 945
HxInstGenConv2dSepMulAdd, 946
HxInstGenConv3dK1dMulAdd, 947
HxInstGenConv3dMulAdd, 947
HxInstGeneratePix, 948
HxInstImpBytes, 949
HxInstImpPackRgb, 949
HxInstImpPpm, 950
HxInstInOutGetPoints, 950
HxInstKerNgb2dNormCorrelation, 951
HxInstNgb2dMean, 952
HxInstNgbIsMaxGradDir2d, 952
HxInstNgbLWshed2d, 953
HxInstNgbNonMaxSuppression2d, 953
HxInstNgbPercentile2d, 954
HxInstRecGenConv2dAddMin, 955
HxInstRecGenConv2dK1dAddMin, 955
HxInstRecGenConv2dK1dMulAdd, 956
HxInstRecGenConv2dMulAdd, 957
HxInstRgb2dBinary, 957
HxInstRgb2dCMY, 958
HxInstRgb2dDirect, 959
HxInstRgb2dDirectNC, 959
HxInstRgb2dHSI, 960
HxInstRgb2dLab, 960
HxInstRgb2dLabel, 961
HxInstRgb2dLogMag, 962
HxInstRgb2dLuv, 962
HxInstRgb2dOOO, 963
HxInstRgb2dStretch, 963
HxInstRgb2dXYZ, 964

HxInstRgb3dBinary, 965
HxInstRgb3dCMY, 965
HxInstRgb3dDirect, 966
HxInstRgb3dHSI, 966
HxInstRgb3dLab, 967
HxInstRgb3dLabel, 968
HxInstRgb3dLogMag, 968
HxInstRgb3dLuv, 969
HxInstRgb3dOOO, 969
HxInstRgb3dStretch, 970
HxInstRgb3dXYZ, 971
HxNgbLocalModeInst, 1052
HxNgbOpticalFlowInst, 1060
f001
HxInstantiatorSet, 912
f002
HxInstantiatorSet, 912
f003
HxInstantiatorSet, 912
f004
HxInstantiatorSet, 912
f005
HxInstantiatorSet, 912
f006
HxInstantiatorSet, 912
f007
HxInstantiatorSet, 912
f008
HxInstantiatorSet, 912
f009
HxInstantiatorSet, 913
f010
HxInstantiatorSet, 913
f011
HxInstantiatorSet, 913
f012
HxInstantiatorSet, 913
f013
HxInstantiatorSet, 913
f014
HxInstantiatorSet, 913
f015
HxInstantiatorSet, 913
f016
HxInstantiatorSet, 913
f017
HxInstantiatorSet, 913
f018
HxInstantiatorSet, 914
f019
HxInstantiatorSet, 914
f020
HxInstantiatorSet, 914
f021
HxInstantiatorSet, 914
falseAlarm
VxStructureEval, 1364
fillNeighborValues
HxImgFtorQueueBased, 823
find
HxImgFtorTable, 857
HxImgFtorTableTem, 859
findKey
HxRegistry, 1092
HxRegKey, 1100
findRelatedBlobs
HxBlob2dRelation, 413
findRelatedBlobsBegin
HxBlob2dRelation, 413
findRelatedBlobsEnd
HxBlob2dRelation, 413
findRelatedBlobsInserter
HxBlob2dRelation, 413
findValue
HxRegistry, 1094
HxRegKey, 1102
first
QThinning, 1361
floor
HxComplex, 516
HxScalarDouble, 1154
HxScalarInt, 1173
HxVec2Double, 1272
HxVec2Int, 1290
HxVec3Double, 1310
HxVec3Int, 1330
found
HxIfRbPair, 581
frame2HxImageRep
HxImageSeqData, 657
HxImageSeqDXMedia, 661
HxImageSeqMDC, 669
frameDepth
HxImageSeq, 651
HxImageSeqData, 656
HxImageSeqDXMedia, 660
HxImageSeqMDC, 668
frameHeight
HxImageSeq, 650
HxImageSeqData, 656
HxImageSeqDXMedia, 659
HxImageSeqMDC, 667
frameWidth
HxImageSeq, 650
HxImageSeqData, 655
HxImageSeqDXMedia, 659
HxImageSeqMDC, 667
from2Images

HxImageFactory, 611
from3Images
HxImageFactory, 611
fromByteData
HxImageFactory, 607
fromDoubleData
HxImageFactory, 609
fromFile
HxImageFactory, 612
fromFloatData
HxImageFactory, 608
fromFunction
HxSFFactory, 1190
fromGenerator
HxImageFactory, 609
fromGrayValue
HxImageFactory, 610
fromImage
HxImageFactory, 607
fromImport
HxImageFactory, 610
fromIntData
HxImageFactory, 608
fromJavaRgb
HxImageFactory, 610
fromMatlab
HxImageFactory, 610
fromNamedGenerator
HxImageFactory, 609
fromShortData
HxImageFactory, 608
fromSignature
HxImageFactory, 607
fromValue
HxImageFactory, 607

genConv2dSep
HxImageData, 597
HxImageRep, 634
genConv3dSep
HxImageData, 598
HxImageRep, 634
genConvSeparated
HxImageData, 596
HxImageRep, 633
generalizedConvolution
HxImageData, 594
HxImageRep, 632
generalizedConvolutionK1d
HxImageData, 595
HxImageRep, 633
geometricOp2d
HxImageData, 604
HxImageRep, 637
HxImageTem2d, 698
HxImageTem3d, 700
get
HxHistogram, 558, 559
getArgument
HxImgFtorKey, 801
getArgumentType
HxImgFtorRuleBase, 844
getAt
HxImageData, 586
HxImageRep, 639
HxImageTem, 695
getBlobBegin
HxBlob2dRelation, 412
HxSegmentation2d, 1185
getBlobEnd
HxBlob2dRelation, 412
HxSegmentation2d, 1185
getBlobInserter
HxSegmentation2d, 1185
getClassName
HxImgFtorKey, 802
HxImgFtorKeyNameTable, 803
getClassNameId
HxImgFtorKeyNameTable, 803
getClosed
HxPolyline2d, 1081
getConnectivity
HxSF, 1186
getCursorKey
HxRegistry, 1094
getCursorName
HxRegistry, 1095
getData
HxRegValue, 1106
getDataDouble
HxHistogram, 572
getDataInt
HxHistogram, 573
getDescription
HxImgFunctor, 863
getDoublePixels
HxImageData, 586
HxImageTem, 695
getExtra2Type
HxImgFtorRuleBase, 846
getExtraType
HxImgFtorRuleBase, 845
getFeature
HxBlob2d, 405
getFeatureInt
HxBlob2d, 405
getFeatureNames
HxBlob2d, 406

getFeatureValue
HxBlob2d, 405
getFilter
DeviceEnumerator, 399
getFrame
HxImageSeq, 651
HxImageSeqData, 656
VideoReader, 1363
getFrameHeight
VideoReader, 1363
getFrameWidth
VideoReader, 1363
getGenerator
HxImageFactory, 606
getHorizontalKernel
HxSF, 1186
getId
HxNameTable, 1037
getInputImage
HxSegmentation2d, 1184
getInt
HxRegData, 1087
HxRegKey, 1102
getInterval
HxBSplineCurve, 482
getIsModifying
HxImgFtorRuleBase, 846
getItemToQueue
QThinning, 1361
getItemToRemove
QThinning, 1361
getItemToWrite
QThinning, 1361
getJidx
HxNJet, 1069
getJList
HxNJet, 1070
getJw
HxNJet, 1071
getKernel
HxSF, 1186
getKernelType
HxImgFtorRuleBase, 845
getKeyList
HxRegKey, 1101
getLabel
HxBlob2d, 404
getLabeledImage
HxSegmentation2d, 1184
getLength
VideoReader, 1363
getLidx
HxNJet, 1069
getList
HxNJet, 1070
getLList
HxNJet, 1070
getLw
HxNJet, 1071
getMidx
HxNJet, 1069
getMList
HxNJet, 1070
getMw
HxNJet, 1071
getName
HxImgFtorKeyNameTable, 803
HxNameTable, 1037
HxRegKey, 1099
HxRegValue, 1106
HxTag, 1194
getNameId
HxImgFtorKeyNameTable, 803
getNames
DeviceEnumerator, 399
HxNameTable, 1037
getNrPoints
HxPolyline2d, 1081
getParent
HxRegKey, 1099
getPoint
HxPolyline2d, 1081
getPoints
HxPolyline2d, 1081, 1082
getPpmPixels
HxImageData, 586
getProjectDomainSizes
HxImageData, 587
getRelation
HxSegmentation2d, 1185
getResultPrecision
HxImageRep, 632
getResultType
HxImgFtorRuleBase, 843
getRgb2d
HxImageSeq, 651
HxImageSeqData, 656
HxImageSeqDXMedia, 660
HxImageSeqMDC, 668
getRgbPixels2d
HxImageData, 586
HxImageRep, 639, 640
HxImageSeq, 651
HxImageSeqData, 657
HxImageSeqDXMedia, 660
HxImageSeqMDC, 668
HxImageTem2d, 698
getRgbPixels3d

HxImageRep, 640
getRootKey
HxRegistry, 1095
getString
HxRegData, 1087
HxRegKey, 1102
getTag
HxTagList, 1198
getTheData
HxHistogram, 571
getTheData2
HxHistogram, 571
getTheData3
HxHistogram, 572
getTypeName
HxImgFtorKeyNameTable, 803
getTypeNameId
HxImgFtorKeyNameTable, 803
getUnshared
HxRcObject, 1083
HxRcPtr, 1084
getValue
HxBlob2dFeatureTem, 409
HxComplex, 512
HxScalarDouble, 1151
HxScalarInt, 1170
HxTagTem, 1203
HxVec2Double, 1269
HxVec2Int, 1287
HxVec3Double, 1307
HxVec3Int, 1327
getValueList
HxRegKey, 1102
getValues
HxImageData, 586
HxImageTem, 695
HxImageTem2d, 697
HxImageTem3d, 699
getVerticalKernel
HxSF, 1186
globalPixelInit
QThinning, 1360

height
HxImageTem2d, 697
HxImageTem3d, 699
highBin
HxHistogram, 557
HxAbs
HxAbs.h, 75
HxAbs.h, 75
HxAbs, 75
HxAcos
HxAcos.h, 76
HxAcos.h, 76
HxAcos, 76
HxAdd
HxAdd.h, 77
HxAdd.h, 77
HxAdd, 77
HxAddBinaryNoise
HxAddBinaryNoise.h, 79
HxAddBinaryNoise.h, 78
HxAddBinaryNoise, 79
HxAddGaussianNoise
HxAddGaussianNoise.h, 79
HxAddGaussianNoise.h, 79
HxAddGaussianNoise, 79
HxAddPoissonNoise
HxAddPoissonNoise.h, 80
HxAddPoissonNoise.h, 80
HxAddPoissonNoise, 80
HxAddSat
HxAddSat.h, 81
HxAddSat.h, 80
HxAddSat, 81
HxAddTag
HxTagList.h, 366
HxAddUniformNoise
HxAddUniformNoise.h, 81
HxAddUniformNoise.h, 81
HxAddUniformNoise, 81
HxAddVal
HxAddVal.h, 82
HxAddVal.h, 82
HxAddVal, 82
HxAffinePix
HxAffinePix.h, 83
HxAffinePix.h
HxAffinePix, 83
HxAffinePix.h, 83
HxAlternateSequentialFilter
HxAlternateSequentialFilter.h, 84
HxAlternateSequentialFilter.h
CO, 83
OC, 83
OCO, 83
opSeq, 83
HxAlternateSequentialFilter.h, 83
HxAlternateSequentialFilter, 84
HxAnd
HxAnd.h, 86
HxAnd.h, 86
HxAnd, 86
HxAndVal
HxAndVal.h, 87
HxAndVal.h, 87
HxAndVal, 87

HxAreaClosing
HxAreaClosing.h, 88
HxAreaClosing.h
HxAreaClosing, 88
HxAreaClosing.h, 88
HxAreaOpening
HxAreaOpening.h, 88
HxAreaOpening.h
HxAreaOpening, 88
HxAreaOpening.h, 88
HxArg
HxArg.h, 89
HxArg.h, 89
HxArg, 89
HxArrowR2
HxArrowR2, 400, 401
HxArrowR2, 400
~HxArrowR2, 401
displace, 401
HxArrowR2, 400, 401
origin, 401
put, 401
HxAsin
HxAsin.h, 90
HxAsin.h, 89
HxAsin, 90
HxAtan
HxAtan.h, 91
HxAtan.h, 91
HxAtan, 91
HxAtan2
HxAtan2.h, 92
HxAtan2.h, 91
HxAtan2, 92
HxBlob2d
HxBlob2d, 403
HxBlob2d, 402
~HxBlob2d, 403
addFeature, 404
getFeature, 405
getFeatureInt, 405
getFeatureNames, 406
getFeatureValue, 405
getLabel, 404
HxBlob2d, 403
ident, 404
put, 406
sizeMaer, 404
startMaer, 404
HxBlob2dFeature
HxBlob2dFeature, 407
HxBlob2dFeature, 406
~HxBlob2dFeature, 407
clone, 408
HxBlob2dFeature, 407
HxBlob2dFeatureTem
HxBlob2dFeatureTem, 409
HxBlob2dFeatureTem, 408
~HxBlob2dFeatureTem, 409
clone, 409
getValue, 409
HxBlob2dFeatureTem, 409
HxBlob2dList, 410
back_insert_iterator, 410
HxBlob2dList.h, 92
HxBlob2dListBackInserter, 93
HxBlob2dListConstIter, 93
HxBlob2dListIter, 93
HxBlob2dListBackInserter
HxBlob2dList.h, 93
HxBlob2dListConstIter
HxBlob2dList.h, 93
HxBlob2dListIter
HxBlob2dList.h, 93
HxBlob2dPtrLess
operator(), 410
HxBlob2dPtrLess, 410
HxBlob2dRelation
HxBlob2dRelation, 412
HxBlob2dRelation, 411
~HxBlob2dRelation, 412
add, 412
findRelatedBlobs, 413
findRelatedBlobsBegin, 413
findRelatedBlobsEnd, 413
findRelatedBlobsInserter, 413
getBlobBegin, 412
getBlobEnd, 412
HxBlob2dRelation, 412
ident, 412
put, 413
HxBoundingBox
HxBoundingBox, 415
HxBoundingBox, 414
begin, 415
end, 416
extend, 416
HxBoundingBox, 415
includes, 417
intersect, 416
isEmpty, 416
put, 417
size, 416
translate, 417
unite, 416
HxBpoAdd
HxBpoAdd, 418
neutralElement, 418

HxBpoAdd, 417
  className, 419
  doIt, 419
  HxBpoAdd, 418
  TransVarianceCategory, 418
HxBpoAddAssign
  ArithType, 419
  HxBpoAddAssign, 420
  neutralElement, 419
HxBpoAddAssign, 419
  className, 420
  doIt, 420
  HxBpoAddAssign, 420
  TransVarianceCategory, 420
HxBpoAddSat
  HxBpoAddSat, 421
HxBpoAddSat, 420
  className, 422
  doIt, 422
  HxBpoAddSat, 421
  TransVarianceCategory, 421
HxBpoAnd
  HxBpoAnd, 423
  neutralElement, 423
HxBpoAnd, 422
  className, 423
  doIt, 423
  HxBpoAnd, 423
  TransVarianceCategory, 423
HxBpoBind2Val
  HxBpoBind2Val, 424
HxBpoBind2Val, 424
  className, 425
  doIt, 425
  HxBpoBind2Val, 424
  TransVarianceCategory, 424
HxBpoCross
  HxBpoCross, 426
HxBpoCross, 425
  className, 426
  doIt, 426
  HxBpoCross, 426
  TransVarianceCategory, 426
HxBpoDiv
  HxBpoDiv, 427
  neutralElement, 427
HxBpoDiv, 426
  className, 428
  doIt, 428
  HxBpoDiv, 427
  TransVarianceCategory, 427
HxBpoDot
  HxBpoDot, 429
HxBpoDot, 428
  className, 429
  doIt, 429
  HxBpoDot, 429
  TransVarianceCategory, 429
HxBpoEqual
  HxBpoEqual, 430
HxBpoEqual, 429
  className, 431
  doIt, 431
  HxBpoEqual, 430
  TransVarianceCategory, 430
HxBpoGreaterEqual
  HxBpoGreaterEqual, 432
HxBpoGreaterEqual, 431
  className, 432
  doIt, 432
  HxBpoGreaterEqual, 432
  TransVarianceCategory, 432
HxBpoGreaterThan
  HxBpoGreaterThan, 433
HxBpoGreaterThan, 432
  className, 434
  doIt, 434
  HxBpoGreaterThan, 433
  TransVarianceCategory, 433
HxBpoHighlightRegion
  HxBpoHighlightRegion, 435
  neutralElement, 434
HxBpoHighlightRegion, 434
  className, 435
  doIt, 435
  HxBpoHighlightRegion, 435
  TransVarianceCategory, 435
HxBpoInf
  HxBpoInf, 436
  neutralElement, 436
HxBpoInf, 436
  className, 437
  doIt, 437
  HxBpoInf, 436
  TransVarianceCategory, 436
HxBpoInfAssign
  ArithType, 437
  HxBpoInfAssign, 438
  neutralElement, 438
HxBpoInfAssign, 437
  className, 438
  doIt, 438
  HxBpoInfAssign, 438
  TransVarianceCategory, 438
HxBpoLeftShift
  HxBpoLeftShift, 439
  neutralElement, 439
HxBpoLeftShift, 439

className, 440
doIt, 440
HxBpoLeftShift, 439
TransVarianceCategory, 439
HxBpoLessEqual
HxBpoLessEqual, 441
HxBpoLessEqual, 440
className, 441
doIt, 441
HxBpoLessEqual, 441
TransVarianceCategory, 441
HxBpoLessThan
HxBpoLessThan, 442
HxBpoLessThan, 441
className, 443
doIt, 443
HxBpoLessThan, 442
TransVarianceCategory, 442
HxBpoMax
HxBpoMax, 444
neutralElement, 443
HxBpoMax, 443
className, 444
doIt, 444
HxBpoMax, 444
TransVarianceCategory, 444
HxBpoMaxAssign
ArithType, 445
HxBpoMaxAssign, 445
neutralElement, 445
HxBpoMaxAssign, 444
className, 446
doIt, 446
HxBpoMaxAssign, 445
TransVarianceCategory, 445
HxBpoMin
HxBpoMin, 447
neutralElement, 446
HxBpoMin, 446
className, 447
doIt, 447
HxBpoMin, 447
TransVarianceCategory, 447
HxBpoMinAssign
ArithType, 448
HxBpoMinAssign, 448
neutralElement, 448
HxBpoMinAssign, 447
className, 449
doIt, 449
HxBpoMinAssign, 448
TransVarianceCategory, 448
HxBpoMod
HxBpoMod, 450
HxBpoMod, 449
className, 450
doIt, 450
HxBpoMod, 450
TransVarianceCategory, 450
HxBpoMul
HxBpoMul, 451
neutralElement, 451
HxBpoMul, 450
className, 452
doIt, 452
HxBpoMul, 451
TransVarianceCategory, 451
HxBpoMulAssign
ArithType, 452
HxBpoMulAssign, 453
neutralElement, 452
HxBpoMulAssign, 452
className, 453
doIt, 453
HxBpoMulAssign, 453
TransVarianceCategory, 453
HxBpoNotEqual
HxBpoNotEqual, 454
HxBpoNotEqual, 453
className, 455
doIt, 455
HxBpoNotEqual, 454
TransVarianceCategory, 454
HxBpoOr
HxBpoOr, 456
neutralElement, 455
HxBpoOr, 455
className, 456
doIt, 456
HxBpoOr, 456
TransVarianceCategory, 456
HxBpoPow
HxBpoPow, 457
HxBpoPow, 456
className, 458
doIt, 458
HxBpoPow, 457
TransVarianceCategory, 457
HxBpoRightShift
HxBpoRightShift, 459
neutralElement, 458
HxBpoRightShift, 458
className, 459
doIt, 459
HxBpoRightShift, 459
TransVarianceCategory, 459
HxBpoSqrDst
HxBpoSqrDst, 460

HxBpoSqrDst, 459
  className, 461
  doIt, 461
  HxBpoSqrDst, 460
  TransVarianceCategory, 460
HxBpoSub
  HxBpoSub, 462
  neutralElement, 461
HxBpoSub, 461
  className, 462
  doIt, 462
  HxBpoSub, 462
  TransVarianceCategory, 462
HxBpoSubAssign
  ArithType, 463
  HxBpoSubAssign, 463
  neutralElement, 463
HxBpoSubAssign, 462
  className, 464
  doIt, 464
  HxBpoSubAssign, 463
  TransVarianceCategory, 463
HxBpoSubSat
  HxBpoSubSat, 465
HxBpoSubSat, 464
  className, 465
  doIt, 465
  HxBpoSubSat, 465
  TransVarianceCategory, 465
HxBpoSup
  HxBpoSup, 467
  neutralElement, 466
HxBpoSup, 466
  className, 467
  doIt, 467
  HxBpoSup, 467
  TransVarianceCategory, 467
HxBpoSupAssign
  ArithType, 467
  HxBpoSupAssign, 468
  neutralElement, 468
HxBpoSupAssign, 467
  className, 468
  doIt, 468
  HxBpoSupAssign, 468
  TransVarianceCategory, 468
HxBpoXor
  HxBpoXor, 470
HxBpoXor, 469
  className, 470
  doIt, 470
  HxBpoXor, 470
  TransVarianceCategory, 470
HxBSplineBasis
  HxBSplineBasis, 472
  HxBSplineCurve, 472
  maxBasis, 472
  nearestKnot, 472
  node, 472
HxBSplineBasis, 470
  ∼HxBSplineBasis, 473
  allKnots, 474
  B, 474
  curveType, 473
  dB, 475
  degree, 473
  dump, 476
  HxBSplineBasis, 472
  insertKnot, 476
  knot, 474
  knotsType, 473
  maxT, 474
  minT, 473
  nIntervals, 473
  numB, 474
  pathAffectedBy, 475
HxBSplineCurve
  HxBSplineBasis, 472
  HxBSplineCurve, 480
HxBSplineCurve, 477
  ∼HxBSplineCurve, 480
  allP, 482
  B, 483
  basis, 481
  C, 484
  center, 486
  changeAllP, 487
  controlP, 482
  curveType, 481
  dB, 483
  dC, 484
  degree, 481
  dTurnAngleAtC, 485
  dump, 490
  getInterval, 482
  HxBSplineCurve, 480
  ident, 481
  insertKnot, 489
  kAtC, 485
  length, 485, 486
  makeInterpolating, 481
  makeUniform, 480
  maxT, 482
  minT, 481
  numP, 482
  P, 482
  pathAffectedBy, 483
  PThatAffectCAt, 483

sampleC, 486
scaleAllP, 487
translateAllP, 487
translateCurve, 487, 488
translateCurve2, 488
HxBSplineInterval
HxBSplineInterval, 492
HxBSplineInterval, 490
~HxBSplineInterval, 492
begin, 492
contains, 493
cropBegin, 494
cropEnd, 494
end, 492
HxBSplineInterval, 492
isClosed, 494
length, 493
middle, 494
next, 493
part, 494
prev, 493
ratio, 494
HxByte
HxByte.h, 93
HxByte.h, 93
HxByte, 93
HxCannyEdgeMap
HxCannyEdgeMap.h, 94
HxCannyEdgeMap.h, 93
HxCannyEdgeMap, 94
HxCannyThreshold
HxCannyThreshold.h, 95
HxCannyThreshold.h, 94
HxCannyThreshold, 95
HxCannyThresholdAlt
HxCannyThresholdAlt.h, 96
HxCannyThresholdAlt.h, 95
HxCannyThresholdAlt, 96
HxCannyThresholdRec
HxCannyThresholdRec.h, 97
HxCannyThresholdRec.h, 96
HxCannyThresholdRec, 97
HxCeil
HxCeil.h, 97
HxCeil.h, 97
HxCeil, 97
HxClassName, 495
operator HxString, 495
HxClosing
HxClosing.h, 98
HxClosing.h
HxClosing, 98
HxClosing.h, 98
HxClosingByReconstruction
HxClosingByReconstruction.h, 98
HxClosingByReconstruction.h, 98
HxClosingByReconstruction, 98
HxClosingByReconstructionTopHat
HxClosingByReconstructionTopHat.h, 99
HxClosingByReconstructionTopHat.h, 99
HxClosingByReconstructionTopHat, 99
HxClosingTopHat
HxClosingTopHat.h, 99
HxClosingTopHat.h, 99
HxClosingTopHat, 99
HxCnum
HxCnum, 496, 497
HxCnum, 496
HxCnum, 496, 497
inc, 498
operator!=, 498
operator=, 497
x, 497
y, 497
z, 498
HxColCMY2RGB
HxColConvert.h, 101
HxColCMY2XYZ
HxColConvert.h, 102
HxColConvert.h, 100
HxColCMY2RGB, 101
HxColCMY2XYZ, 102
HxColHSI2RGB, 105
HxColLab2XYZ, 102
HxColLuv2XYZ, 103
HxColOOO2RGB, 104
HxColOOO2XYZ, 104
HxColRGB2CMY, 101
HxColRGB2HSI, 105
HxColRGB2int, 106
HxColRGB2OOO, 104
HxColRGB2XYZ, 101
HxColXYZ2CMY, 102
HxColXYZ2Lab, 103
HxColXYZ2Luv, 103
HxColXYZ2OOO, 104
HxColXYZ2RGB, 101
HxColHSI2RGB
HxColConvert.h, 105
HxColLab2XYZ
HxColConvert.h, 102
HxColLuv2XYZ
HxColConvert.h, 103
HxColOOO2RGB
HxColConvert.h, 104
HxColOOO2XYZ
HxColConvert.h, 104
HxColor

HxColor, 500
HxColor, 498
convert, 500
HxColor, 500
operator!=, 505
operator=, 500
operator==, 505
put, 505
toCMY, 501
toHSI, 504
toLab, 502
toLuv, 503
toOOO, 503
toRGB, 501
toString, 505
toXYZ, 502
value, 500
HxColor.h
operator<<, 107
HxColor.h, 106
HxColorModel, 107
HxColorModel
HxColor.h, 107
HxColorSpace
HxColorSpace.h, 107
HxColorSpace.h, 107
HxColorSpace, 107
HxColRGB2CMY
HxColConvert.h, 101
HxColRGB2HSI
HxColConvert.h, 105
HxColRGB2int
HxColConvert.h, 106
HxColRGB2OOO
HxColConvert.h, 104
HxColRGB2XYZ
HxColConvert.h, 101
HxColXYZ2CMY
HxColConvert.h, 102
HxColXYZ2Lab
HxColConvert.h, 103
HxColXYZ2Luv
HxColConvert.h, 103
HxColXYZ2OOO
HxColConvert.h, 104
HxColXYZ2RGB
HxColConvert.h, 101
HxComplement
HxComplement.h, 108
HxComplement.h, 108
HxComplement, 108
HxComplex
HxComplex, 512
operator new, 511
operator=, 511
setValue, 507
HxComplex, 506
abs, 515
acos, 518
and, 523
arg, 515
asin, 518
atan, 518
atan2, 519
ceil, 515
complement, 515
conjugate, 515
cos, 518
cosh, 519
cross, 524
dim, 512
dot, 524
exp, 520
floor, 516
getValue, 512
HxComplex, 512
inf, 522
infAssign, 522
LARGE_VAL, 526
leftShift, 524
log, 520
log10, 520
max, 516, 522
maxAssign, 522
min, 516, 521
minAssign, 521
mod, 523
norm1, 517
norm2, 517
normInf, 517
operator *, 525
operator *=, 521
operator HxScalarDouble, 513
operator HxScalarInt, 513
operator HxVec2Double, 513
operator HxVec2Int, 513
operator HxVec3Double, 513
operator HxVec3Int, 513
operator!=, 514
operator+, 525
operator+=, 520
operator-, 515, 525
operator-=, 520
operator/, 525
operator/=, 521
operator<, 514
operator<=, 514
operator==, 514

operator>, 514
operator>=, 514
or, 523
pow, 523
product, 516
put, 524
rightShift, 524
round, 516
sin, 517
sinh, 519
SMALL_VAL, 526
sqrt, 517
sum, 516
sup, 522
supAssign, 522
tan, 518
tanh, 519
toString, 524
x, 512
xor, 523
y, 512
HxComplexValue
HxValue, 1258
HxConditionalDilation
HxConditionalDilation.h, 109
HxConditionalDilation.h, 109
HxConditionalDilation, 109
HxConditionalErosion
HxConditionalErosion.h, 110
HxConditionalErosion.h, 110
HxConditionalErosion, 110
HxConjugate
HxConjugate.h, 111
HxConjugate.h, 111
HxConjugate, 111
HxContrastStretch
HxContrastStretch.h, 112
HxContrastStretch.h, 112
HxContrastStretch, 112
HxConvGauss2d
HxConvGauss2d.h, 113
HxConvGauss2d.h, 112
HxConvGauss2d, 113
HxConvGauss3d
HxConvGauss3d.h, 114
HxConvGauss3d.h, 114
HxConvGauss3d, 114
HxConvKernelSeparated
HxConvKernelSeparated.h, 116
HxConvKernelSeparated.h, 116
HxConvKernelSeparated, 116
HxConvKernelSeparated2d
HxConvKernelSeparated2d.h, 117
HxConvKernelSeparated2d.h
HxConvKernelSeparated2d, 117
HxConvKernelSeparated2d.h, 117
HxConvKernelSeparated2d, 117
HxConvolution
HxConvolution.h, 118
HxConvolution.h, 117
HxConvolution, 118
HxCoord
x, 526
y, 526
z, 526
HxCoord, 526
HxCos
HxCos.h, 119
HxCos.h, 118
HxCos, 119
HxCosh
HxCosh.h, 119
HxCosh.h, 119
HxCosh, 119
HxCross
HxCross.h, 120
HxCross.h, 120
HxCross, 120
HxCrossVal
HxCrossVal.h, 121
HxCrossVal.h, 121
HxCrossVal, 121
HxDataPtr2dScalarTem
data, 527
decX, 527
decXYZ, 527
decY, 527
decZ, 527
HxDataPtr2dScalarTem, 526
incX, 527
incXYZ, 527
incY, 527
incZ, 527
operator=, 527
read, 527
readIncX, 527
vprint, 527
write, 527
writeIncX, 527
HxDataPtr2dScalarTem, 526
HxDataPtr2dTem
data, 528
decX, 527, 528
decXYZ, 528
decY, 527, 528
decZ, 528
HxDataPtr2dTem, 527
incX, 527, 528

incXYZ, 528
incY, 527, 528
incZ, 527
operator=, 527
read, 528
readIncX, 528
vprint, 528
write, 528
writeIncX, 528
HxDataPtr2dTem, 527
HxDataPtr3dScalarTem
data, 529
decX, 528
decXYZ, 529
decY, 528
decZ, 528, 529
HxDataPtr3dScalarTem, 528
incX, 528
incXYZ, 529
incY, 528
incZ, 528
operator=, 528
read, 529
readIncX, 529
vprint, 529
write, 529
writeIncX, 529
HxDataPtr3dScalarTem, 528
HxDefuz
HxDefuz.h, 122
HxDefuz.h, 122
HxDefuz, 122
HxDilation
HxDilation.h, 123
HxDilation.h
HxDilation, 123
HxDilation.h, 123
HxDisplayOF
HxDisplayOF.h, 123
HxDisplayOF.h, 123
HxDisplayOF, 123
HxDistanceTransform
HxDistanceTransform.h, 125
HxDistanceTransform.h, 124
HxDistanceTransform, 125
HxDistanceTransformMM
HxDistanceTransformMM.h, 125
HxDistanceTransformMM.h, 125
HxDistanceTransformMM, 125
HxDiv
HxDiv.h, 126
HxDiv.h, 126
HxDiv, 126
HxDivVal
HxDivVal.h, 128
HxDivVal.h, 127
HxDivVal, 128
HxDiyTranspose
HxDiyTranspose, 530
HxDiyTranspose, 529
className, 530
doIt, 530
HxDiyTranspose, 530
HxDot
HxDot.h, 130
HxDot.h, 130
HxDot, 130
HxDotVal
HxDotVal.h, 132
HxDotVal.h, 131
HxDotVal, 132
HxEqual
HxEqual.h, 133
HxEqual.h, 132
HxEqual, 133
HxEqualVal
HxEqualVal.h, 134
HxEqualVal.h, 134
HxEqualVal, 134
HxErosion
HxErosion.h, 135
HxErosion.h
HxErosion, 135
HxErosion.h, 135
HxExp
HxExp.h, 135
HxExp.h, 135
HxExp, 135
HxExportByteData
HxExportByteData.h, 136
HxExportByteData.h, 135
HxExportByteData, 136
HxExportDoubleData
HxExportDoubleData.h, 136
HxExportDoubleData.h, 136
HxExportDoubleData, 136
HxExportExtraIdentMaskCentralMoments
HxExportExtraIdentMaskCentralMoments, 532
HxExportExtraIdentMaskCentralMoments, 530
~HxExportExtraIdentMaskCentralMoments, 532
className, 534
doIt, 533
done, 533
HxExportExtraIdentMaskCentralMoments, 532
init, 533

nrPhases, 533
PhaseCategory, 532
TransVarianceCategory, 532
HxExportExtraIdentMaskMean
HxExportExtraIdentMaskMean, 535
HxExportExtraIdentMaskMean, 534
~HxExportExtraIdentMaskMean, 535
className, 536
doIt, 536
HxExportExtraIdentMaskMean, 535
PhaseCategory, 535
TransVarianceCategory, 535
HxExportExtraIdentMaskMedian
HxExportExtraIdentMaskMedian, 537
HxExportExtraIdentMaskMedian, 536
~HxExportExtraIdentMaskMedian, 537
className, 538
doIt, 538
HxExportExtraIdentMaskMedian, 537
PhaseCategory, 537
TransVarianceCategory, 537
HxExportExtraIdentMaskMoments
HxExportExtraIdentMaskMoments, 540
HxExportExtraIdentMaskMoments, 538
~HxExportExtraIdentMaskMoments, 540
className, 541
doIt, 540
HxExportExtraIdentMaskMoments, 540
PhaseCategory, 540
TransVarianceCategory, 540
HxExportExtraIdentMaskStdev
HxExportExtraIdentMaskStdev, 542
HxExportExtraIdentMaskStdev, 541
~HxExportExtraIdentMaskStdev, 542
className, 543
doIt, 543
HxExportExtraIdentMaskStdev, 542
PhaseCategory, 542
TransVarianceCategory, 542
HxExportExtraIdentMaskSum
HxExportExtraIdentMaskSum, 545
HxExportExtraIdentMaskSum, 543
~HxExportExtraIdentMaskSum, 545
className, 545
doIt, 545
HxExportExtraIdentMaskSum, 545
PhaseCategory, 544
TransVarianceCategory, 544
HxExportExtraWeightMaskSum
HxExportExtraWeightMaskSum, 547
HxExportExtraWeightMaskSum, 546
~HxExportExtraWeightMaskSum, 547
className, 547
doIt, 547
HxExportExtraWeightMaskSum, 547
PhaseCategory, 546
TransVarianceCategory, 546
HxExportFloatData
HxExportFloatData.h, 137
HxExportFloatData.h, 137
HxExportFloatData, 137
HxExportIntData
HxExportIntData.h, 138
HxExportIntData.h, 138
HxExportIntData, 138
HxExportMatlabPixels
HxExportMatlabPixels.h, 139
HxImageRep, 646
HxExportMatlabPixels.h, 139
HxExportMatlabPixels, 139
HxExportPpmPixels
HxExportPpmPixels.h, 139
HxExportPpmPixels.h
HxExportPpmPixels, 139
HxExportPpmPixels.h, 139
HxExportShortData
HxExportShortData.h, 140
HxExportShortData.h, 140
HxExportShortData, 140
HxExtend
HxExtend.h, 141
HxImageRep, 644
HxExtend.h, 140
HxExtend, 141
HxExtendVal
HxExtendVal.h, 142
HxImageRep, 645
HxExtendVal.h, 142
HxExtendVal, 142
HxFloor
HxFloor.h, 144
HxFloor.h, 143
HxFloor, 144
HxFuncBorderConstant2d
HxFuncBorderOp.h, 147
HxFuncBorderConstant3d
HxFuncBorderOp.h, 148
HxFuncBorderMirror2d
HxFuncBorderOp.h, 145
HxFuncBorderMirror3d
HxFuncBorderOp.h, 146
HxFuncBorderOp.h, 144
HxFuncBorderConstant2d, 147
HxFuncBorderConstant3d, 148
HxFuncBorderMirror2d, 145
HxFuncBorderMirror3d, 146
HxFuncBorderPropagate2d, 149
HxFuncBorderPropagate3d, 150

HxFuncBorderPropagate2d
HxFuncBorderOp.h, 149
HxFuncBorderPropagate3d
HxFuncBorderOp.h, 150
HxFuncBpo
HxFuncBpo.c, 152
HxFuncBpo.c, 152
HxFuncBpo, 152
HxFuncBpoDispatch, 153
HxFuncBpoDispatch
HxFuncBpo.c, 153
HxFuncExportExtra
HxFuncExportExtra.c, 154–156
HxFuncExportExtra.c, 153
HxFuncExportExtra, 154–156
HxFuncExportExtra_Row_OutTi, 154
HxFuncExportExtra_Row_OutTv, 154
HxFuncExportExtraDispatch, 156
HxFuncExportExtra_Row_OutTi
HxFuncExportExtra.c, 154
HxFuncExportExtra_Row_OutTv
HxFuncExportExtra.c, 154
HxFuncExportExtraDispatch
HxFuncExportExtra.c, 156
HxFuncGenConv2d.c
HxFuncGenConv2d_norowpixfunc, 157
HxFuncGenConv2d_rowpixfunc, 157
HxFuncGenConv2d.c, 156
HxFuncGenConv2dDispatch, 157
HxFuncGenConv2d_norowpixfunc
HxFuncGenConv2d.c, 157
HxFuncGenConv2d_rowpixfunc
HxFuncGenConv2d.c, 157
HxFuncGenConv2dDispatch
HxFuncGenConv2d.c, 157
HxFuncGenConv2dK1d.c, 157
HxFuncGenConv2dK1d_Im_XdirInp, 160
HxFuncGenConv2dK1d_Im_YdirInp, 161
HxFuncGenConv2dK1d_Line_XdirInp, 159
HxFuncGenConv2dK1d_Line_XdirSim, 160
HxFuncGenConv2dK1d_Line_YdirInp, 159
HxFuncGenConv2dK1d_Line_YdirSim, 160
HxFuncGenConv2dK1d_XdirInp, 163
HxFuncGenConv2dK1d_XdirSim, 162
HxFuncGenConv2dK1d_YdirInp, 163
HxFuncGenConv2dK1d_YdirSim, 162
HxFuncGenConv2dK1dDispatch, 164
HxFuncGenConv2dK1d_Im_XdirInp
HxFuncGenConv2dK1d.c, 160
HxFuncGenConv2dK1d_Im_YdirInp
HxFuncGenConv2dK1d.c, 161
HxFuncGenConv2dK1d_Line_XdirInp
HxFuncGenConv2dK1d.c, 159
HxFuncGenConv2dK1d_Line_XdirSim
HxFuncGenConv2dK1d.c, 160
HxFuncGenConv2dK1d_Line_YdirInp
HxFuncGenConv2dK1d.c, 159
HxFuncGenConv2dK1d_Line_YdirSim
HxFuncGenConv2dK1d.c, 160
HxFuncGenConv2dK1d_XdirInp
HxFuncGenConv2dK1d.c, 163
HxFuncGenConv2dK1d_XdirSim
HxFuncGenConv2dK1d.c, 162
HxFuncGenConv2dK1d_YdirInp
HxFuncGenConv2dK1d.c, 163
HxFuncGenConv2dK1d_YdirSim
HxFuncGenConv2dK1d.c, 162
HxFuncGenConv2dK1dDispatch
HxFuncGenConv2dK1d.c, 164
HxFuncGenConv2dSep.c, 164
HxFuncGenConv2dSep_CopyKernel, 173
HxFuncGenConv2dSep_Hor, 174
HxFuncGenConv2dSep_Line_Xdir, 168
HxFuncGenConv2dSep_Line_XdirInc, 168
HxFuncGenConv2dSep_Line_XdirVerInc, 168
HxFuncGenConv2dSep_Line_-XYdirMinInc, 172
HxFuncGenConv2dSep_Line_-XYdirVerCycInc, 172
HxFuncGenConv2dSep_Line_YdirHor, 170
HxFuncGenConv2dSep_Line_YdirHorInc, 170
HxFuncGenConv2dSep_Line_YdirNaiInc, 169
HxFuncGenConv2dSep_Line_YdirSim, 169
HxFuncGenConv2dSep_Line_YdirSimInc, 170
HxFuncGenConv2dSep_Line_YdirVerInc, 171
HxFuncGenConv2dSep_Min, 179
HxFuncGenConv2dSep_Pix_Xdir, 167
HxFuncGenConv2dSep_Pix_Ydir, 167
HxFuncGenConv2dSep_Sim, 173
HxFuncGenConv2dSep_Ver, 176
HxFuncGenConv2dSep_VerCyc, 177
HxFuncGenConv2dSepDispatch, 180
HxFuncGenConv2dSep_CopyKernel
HxFuncGenConv2dSep.c, 173
HxFuncGenConv2dSep_Hor
HxFuncGenConv2dSep.c, 174
HxFuncGenConv2dSep_Line_Xdir
HxFuncGenConv2dSep.c, 168
HxFuncGenConv2dSep_Line_XdirInc
HxFuncGenConv2dSep.c, 168

HxFuncGenConv2dSep_Line_XdirVerInc
HxFuncGenConv2dSep.c, 168
HxFuncGenConv2dSep_Line_XYdirMinInc
HxFuncGenConv2dSep.c, 172
HxFuncGenConv2dSep_Line_XYdirVerCycInc
HxFuncGenConv2dSep.c, 172
HxFuncGenConv2dSep_Line_YdirHor
HxFuncGenConv2dSep.c, 170
HxFuncGenConv2dSep_Line_YdirHorInc
HxFuncGenConv2dSep.c, 170
HxFuncGenConv2dSep_Line_YdirNaiInc
HxFuncGenConv2dSep.c, 169
HxFuncGenConv2dSep_Line_YdirSim
HxFuncGenConv2dSep.c, 169
HxFuncGenConv2dSep_Line_YdirSimInc
HxFuncGenConv2dSep.c, 170
HxFuncGenConv2dSep_Line_YdirVerInc
HxFuncGenConv2dSep.c, 171
HxFuncGenConv2dSep_Min
HxFuncGenConv2dSep.c, 179
HxFuncGenConv2dSep_Pix_Xdir
HxFuncGenConv2dSep.c, 167
HxFuncGenConv2dSep_Pix_Ydir
HxFuncGenConv2dSep.c, 167
HxFuncGenConv2dSep_Sim
HxFuncGenConv2dSep.c, 173
HxFuncGenConv2dSep_Ver
HxFuncGenConv2dSep.c, 176
HxFuncGenConv2dSep_VerCyc
HxFuncGenConv2dSep.c, 177
HxFuncGenConv2dSepDispatch
HxFuncGenConv2dSep.c, 180
HxFuncGenConv3d.c
HxFuncGenConv3d_norowpixfunc, 181
HxFuncGenConv3d_rowpixfunc, 181
HxFuncGenConv3d.c, 181
HxFuncGenConv3dDispatch, 181
HxFuncGenConv3d_norowpixfunc
HxFuncGenConv3d.c, 181
HxFuncGenConv3d_rowpixfunc
HxFuncGenConv3d.c, 181
HxFuncGenConv3dDispatch
HxFuncGenConv3d.c, 181
HxFuncGenConv3dK1d.c, 181
HxFuncGenConv3dK1d_XdirSim, 182
HxFuncGenConv3dK1d_YdirSim, 183
HxFuncGenConv3dK1d_ZdirSim, 183
HxFuncGenConv3dK1dDispatch, 184
HxFuncGenConv3dK1d_XdirSim
HxFuncGenConv3dK1d.c, 182
HxFuncGenConv3dK1d_YdirSim
HxFuncGenConv3dK1d.c, 183
HxFuncGenConv3dK1d_ZdirSim
HxFuncGenConv3dK1d.c, 183
HxFuncGenConv3dK1dDispatch
HxFuncGenConv3dK1d.c, 184
HxFuncInOut
HxFuncInOut.c, 187–189
HxFuncInOut.c, 185
HxFuncInOut, 187–189
HxFuncInOut_Row_InTi, 186
HxFuncInOut_Row_InTv, 186
HxFuncInOut_Row_OutTi, 186
HxFuncInOut_Row_OutTv, 187
HxFuncInOutDispatch, 190
HxFuncInOut_Row_InTi
HxFuncInOut.c, 186
HxFuncInOut_Row_InTv
HxFuncInOut.c, 186
HxFuncInOut_Row_OutTi
HxFuncInOut.c, 186
HxFuncInOut_Row_OutTv
HxFuncInOut.c, 187
HxFuncInOutDispatch
HxFuncInOut.c, 190
HxFuncInOutInit
HxFuncInOutInit.h, 191, 192
HxFuncInOutInit.h, 190
HxFuncInOutInit, 191, 192
HxFuncKernelNgbOp2d.c, 193
HxFuncKernelNgbOp2d_Pix_P1Loop, 194
HxFuncKernelNgbOp2d_Pix_P2Loop, 194
HxFuncKernelNgbOp2d_Row, 194, 195
HxFuncKernelNgbOp2dDispatch, 195
HxFuncKernelNgbOp2d_Pix_P1Loop
HxFuncKernelNgbOp2d.c, 194
HxFuncKernelNgbOp2d_Pix_P2Loop
HxFuncKernelNgbOp2d.c, 194
HxFuncKernelNgbOp2d_Row
HxFuncKernelNgbOp2d.c, 194, 195
HxFuncKernelNgbOp2dDispatch
HxFuncKernelNgbOp2d.c, 195
HxFuncMNpo
HxFuncMNpo.c, 196
HxFuncMNpo.c, 196
HxFuncMNpo, 196
HxFuncMNpoDispatch, 197
HxFuncMNpoDispatch
HxFuncMNpo.c, 197
HxFuncMpo
HxFuncMpo.c, 198
HxFuncMpo.c, 197
HxFuncMpo, 198
HxFuncMpoDispatch, 198
HxFuncMpoDispatch
HxFuncMpo.c, 198
HxFuncNgbOp2d.c, 198
HxFuncNgbOp2d_Pix_P1Cnum, 199

HxFuncNgbOp2d_Pix_P1Loop, 200
HxFuncNgbOp2d_Pix_P2Loop, 200
HxFuncNgbOp2d_Row, 200, 201
HxFuncNgbOp2dDispatch, 202
HxFuncNgbOp2d_Pix_P1Cnum
HxFuncNgbOp2d.c, 199
HxFuncNgbOp2d_Pix_P1Loop
HxFuncNgbOp2d.c, 200
HxFuncNgbOp2d_Pix_P2Loop
HxFuncNgbOp2d.c, 200
HxFuncNgbOp2d_Row
HxFuncNgbOp2d.c, 200, 201
HxFuncNgbOp2dDispatch
HxFuncNgbOp2d.c, 202
HxFuncNgbOp2dExtra.c, 202
HxFuncNgbOp2dExtra_Pix_P1Cnum, 203
HxFuncNgbOp2dExtra_Pix_P1Loop, 203
HxFuncNgbOp2dExtra_Pix_P2Loop, 204
HxFuncNgbOp2dExtra_Row, 204, 205
HxFuncNgbOp2dExtraDispatch, 206
HxFuncNgbOp2dExtra2.c, 206
HxFuncNgbOp2dExtra2_Pix_P1Cnum, 208
HxFuncNgbOp2dExtra2_Pix_P1Loop, 208
HxFuncNgbOp2dExtra2_Pix_P2Loop, 208
HxFuncNgbOp2dExtra2_Row, 209, 210
HxFuncNgbOp2dExtra2Dispatch, 211
HxFuncNgbOp2dExtra2_Pix_P1Cnum
HxFuncNgbOp2dExtra2.c, 208
HxFuncNgbOp2dExtra2_Pix_P1Loop
HxFuncNgbOp2dExtra2.c, 208
HxFuncNgbOp2dExtra2_Pix_P2Loop
HxFuncNgbOp2dExtra2.c, 208
HxFuncNgbOp2dExtra2_Row
HxFuncNgbOp2dExtra2.c, 209, 210
HxFuncNgbOp2dExtra2Dispatch
HxFuncNgbOp2dExtra2.c, 211
HxFuncNgbOp2dExtra_Pix_P1Cnum
HxFuncNgbOp2dExtra.c, 203
HxFuncNgbOp2dExtra_Pix_P1Loop
HxFuncNgbOp2dExtra.c, 203
HxFuncNgbOp2dExtra_Pix_P2Loop
HxFuncNgbOp2dExtra.c, 204
HxFuncNgbOp2dExtra_Row
HxFuncNgbOp2dExtra.c, 204, 205
HxFuncNgbOp2dExtraDispatch
HxFuncNgbOp2dExtra.c, 206
HxFuncRecGenConv2d.c, 211
HxFuncRecGenConv2d_XYdirSim, 212
HxFuncRecGenConv2dDispatch, 213
HxFuncRecGenConv2d_XYdirSim
HxFuncRecGenConv2d.c, 212
HxFuncRecGenConv2dDispatch
HxFuncRecGenConv2d.c, 213
HxFuncRecGenConv2dK1d.c
HxFuncRecGenConv2dK1d_Line_YdirBufAnti, 214
HxFuncRecGenConv2dK1d.c, 213
HxFuncRecGenConv2dK1d_Line_XdirSim, 215
HxFuncRecGenConv2dK1d_Line_YdirBuf, 216
HxFuncRecGenConv2dK1d_Line_YdirSim, 215
HxFuncRecGenConv2dK1d_XdirSim, 217
HxFuncRecGenConv2dK1d_YdirBuf, 218
HxFuncRecGenConv2dK1d_YdirSim, 217
HxFuncRecGenConv2dK1dDispatch, 218
HxFuncRecGenConv2dK1d_Line_XdirSim
HxFuncRecGenConv2dK1d.c, 215
HxFuncRecGenConv2dK1d_Line_YdirBuf
HxFuncRecGenConv2dK1d.c, 216
HxFuncRecGenConv2dK1d_Line_YdirBufAnti
HxFuncRecGenConv2dK1d.c, 214
HxFuncRecGenConv2dK1d_Line_YdirSim
HxFuncRecGenConv2dK1d.c, 215
HxFuncRecGenConv2dK1d_XdirSim
HxFuncRecGenConv2dK1d.c, 217
HxFuncRecGenConv2dK1d_YdirBuf
HxFuncRecGenConv2dK1d.c, 218
HxFuncRecGenConv2dK1d_YdirSim
HxFuncRecGenConv2dK1d.c, 217
HxFuncRecGenConv2dK1dDispatch
HxFuncRecGenConv2dK1d.c, 218
HxFuncRgbOp2d
HxFuncRgbOp2d.c, 220
HxFuncRgbOp2d.c, 219
HxFuncRgbOp2d, 220
HxFuncRgbOp3d
HxFuncRgbOp3d.c, 221
HxFuncRgbOp3d.c, 220
HxFuncRgbOp3d, 221
HxFuncSet
HxFuncSet.c, 223
HxFuncSet.c
DataPtr2dDouble, 222
DataPtr2dVec2Double, 222
HxFuncSet.c, 222
HxFuncSet, 223
HxFuncSet_Row, 223
HxFuncSet_Row< DataPtr2dDouble, DataPtr2dDouble >, 223
HxFuncSet_Row< DataPtr2dVec2Double, DataPtr2dVec2Double >, 223
HxFuncSet_Row
HxFuncSet.c, 223
HxFuncSet_Row< DataPtr2dDouble, DataPtr2dDouble >
HxFuncSet.c, 223

HxFuncSet_Row< DataPtr2dVec2Double, DataPtr2dVec2Double >
HxFuncSet.c, 223
HxFuncUpo
HxFuncUpo.c, 224
HxFuncUpo.c, 224
HxFuncUpo, 224
HxFuncUpoDispatch, 225
HxFuncUpoDispatch
HxFuncUpo.c, 225
HxGauss
HxGauss.h, 225
HxGauss.h, 225
HxGauss, 225
HxGaussDerivative2d
HxGaussDerivative2d.h, 226
HxGaussDerivative2d.h, 226
HxGaussDerivative2d, 226
HxGaussDerivative3d
HxGaussDerivative3d.h, 228
HxGaussDerivative3d.h, 227
HxGaussDerivative3d, 228
HxGaussianDeblur
HxGaussianDeblur.h, 229
HxGaussianDeblur.h, 229
HxGaussianDeblur, 229
HxGeodesicDistanceTransform
HxGeodesicDistanceTransform.h, 230
HxGeodesicDistanceTransform.h, 230
HxGeodesicDistanceTransform, 230
HxGeoIntType
HxGeoIntType.h, 231
HxGeoIntType.h
HxGeoIntType_put, 231
makeString, 231
operator<<, 231
HxGeoIntType.h, 230
HxGeoIntType, 231
HxGeoIntType_put
HxGeoIntType.h, 231
HxGetBlobFeatures
HxGetBlobFeatures.h, 231
HxGetBlobFeatures.h
BasicFeatures, 231
BasicFeaturesList, 231
HxGetBlobFeatures, 231
HxGetBlobFeatures.h, 231
HxGetPoints
HxGetPoints.h, 232
HxGetPoints.h
HxGetPoints, 232
HxGetPoints.h, 232
HxGetTag
HxTagList.h, 366
HxGetValues
HxGetValues.h, 232
HxImageRep, 625
HxGetValues.h
HxGetValues, 232
HxGetValues.h, 232
HxGreaterEqual
HxGreaterEqual.h, 232
HxGreaterEqual.h, 232
HxGreaterEqual, 232
HxGreaterEqualVal
HxGreaterEqualVal.h, 234
HxGreaterEqualVal.h, 233
HxGreaterEqualVal, 234
HxGreaterThan
HxGreaterThan.h, 234
HxGreaterThan.h, 234
HxGreaterThan, 234
HxGreaterThanVal
HxGreaterThanVal.h, 236
HxGreaterThanVal.h, 235
HxGreaterThanVal, 236
HxHighlightRegion
HxHighlightRegion.h, 237
HxHighlightRegion.h, 236
HxHighlightRegion, 237
HxHilditchSkeleton
HxHilditchSkeleton.h, 237
HxHilditchSkeleton.h, 237
HxHilditchSkeleton, 237
HxHistogram
computeEntropyThreshold, 552
computeIsodataThreshold, 552
HxHistogram, 553–555
HxHistogram, 548
~HxHistogram, 555
binToValue, 558
binWidth, 557
chiSquare, 570
chiSquareNorm, 570
convert, 571
countBins, 576
dataType, 556
dimensionality, 556
dimensionSize, 556
get, 558, 559
getDataDouble, 572
getDataInt, 573
getTheData, 571
getTheData2, 571
getTheData3, 572
highBin, 557
HxHistogram, 553–555
ident, 556

incBin, 564
incBinChecked, 560, 561
insertVal, 561–563
insertValChecked, 559, 560
intersection, 569
isNull, 555
lowBin, 557
maxVal, 568, 569
minVal, 568
modes, 566
normalize, 567
nrOfBins, 557
operator int, 556
operator=, 555
put, 574
reduceRange, 577
reduceRangeVal, 578
render3d, 573
setBin, 564, 565
smooth, 566
sum, 567
threshold, 576
to1D, 579
valueToBin, 558
write, 575
HxHitOrMiss
HxHitOrMiss.h, 238
HxHitOrMiss.h, 238
HxHitOrMiss, 238
HxIdentMaskCentralMoments
HxIdentMaskCentralMoments.h, 238
HxIdentMaskCentralMoments.h, 238
HxIdentMaskCentralMoments, 238
HxIdentMaskMean
HxIdentMaskMean.h, 239
HxIdentMaskMean.h, 239
HxIdentMaskMean, 239
HxIdentMaskMedian
HxIdentMaskMedian.h, 240
HxIdentMaskMedian.h, 240
HxIdentMaskMedian, 240
HxIdentMaskMoments
HxIdentMaskMoments.h, 241
HxIdentMaskMoments.h, 240
HxIdentMaskMoments, 241
HxIdentMaskStDev
HxIdentMaskStDev.h, 242
HxIdentMaskStDev.h, 241
HxIdentMaskStDev, 242
HxIdentMaskSum
HxIdentMaskSum.h, 242
HxIdentMaskSum.h, 242
HxIdentMaskSum, 242
HxIdentMaskVariance
HxIdentMaskVariance.h, 243
HxIdentMaskVariance.h, 243
HxIdentMaskVariance, 243
HxIfRbPair
HxIfRbPair, 581
HxIfRbPair, 580
found, 581
HxIfRbPair, 581
operator HxImageSignature, 581
operator int, 581
sig, 581
HxImageAsByte
HxImageAsByte.h, 244
HxImageAsByte.h, 244
HxImageAsByte, 244
HxImageAsComplex
HxImageAsComplex.h, 245
HxImageAsComplex.h, 244
HxImageAsComplex, 245
HxImageAsDouble
HxImageAsDouble.h, 245
HxImageAsDouble.h, 245
HxImageAsDouble, 245
HxImageAsFloat
HxImageAsFloat.h, 246
HxImageAsFloat.h, 246
HxImageAsFloat, 246
HxImageAsInt
HxImageAsInt.h, 247
HxImageAsInt.h, 247
HxImageAsInt, 247
HxImageAsShort
HxImageAsShort.h, 248
HxImageAsShort.h, 247
HxImageAsShort, 248
HxImageAsVec2Byte
HxImageAsVec2Byte.h, 248
HxImageAsVec2Byte.h, 248
HxImageAsVec2Byte, 248
HxImageAsVec2Double
HxImageAsVec2Double.h, 249
HxImageAsVec2Double.h, 249
HxImageAsVec2Double, 249
HxImageAsVec2Float
HxImageAsVec2Float.h, 250
HxImageAsVec2Float.h, 250
HxImageAsVec2Float, 250
HxImageAsVec2Int
HxImageAsVec2Int.h, 251
HxImageAsVec2Int.h, 250
HxImageAsVec2Int, 251
HxImageAsVec2Short
HxImageAsVec2Short.h, 251
HxImageAsVec2Short.h, 251

HxImageAsVec2Short, 251
HxImageAsVec3Byte
HxImageAsVec3Byte.h, 252
HxImageAsVec3Byte.h, 252
HxImageAsVec3Byte, 252
HxImageAsVec3Double
HxImageAsVec3Double.h, 253
HxImageAsVec3Double.h, 253
HxImageAsVec3Double, 253
HxImageAsVec3Float
HxImageAsVec3Float.h, 254
HxImageAsVec3Float.h, 253
HxImageAsVec3Float, 254
HxImageAsVec3Int
HxImageAsVec3Int.h, 254
HxImageAsVec3Int.h, 254
HxImageAsVec3Int, 254
HxImageAsVec3Short
HxImageAsVec3Short.h, 255
HxImageAsVec3Short.h, 255
HxImageAsVec3Short, 255
HxImageData
checkBorderSize, 587
checkEqualImageSig, 587
checkEqualImageSigAndSizes, 587
checkEqualImageSizes, 587
checkEqualImageSizesDim, 587
checkImageDimension, 587
checkLargerImageSigAndSizes, 587
checkPixelDimension, 587
checkProperKernelSigAndSizes, 587
clone, 586
extend, 586
getAt, 586
getDoublePixels, 586
getPpmPixels, 586
getProjectDomainSizes, 587
getRgbPixels2d, 586
getValues, 586
HxImageData, 587
inverseProjectDomain, 586
inverseProjectRange, 586
mirrorBorder, 584
printInfo, 586
probeMNpo, 586
projectDomain, 586
projectRange, 586
propagateBorder, 584
restrict, 586
set, 584
setAt, 586
setBorder, 584
setObjectObserver, 586
setPartImage, 584
setPpmPixels, 586
weight, 586
HxImageData, 581
∼HxImageData, 587
binaryPixOp, 593
dimensionality, 588
dimensionSize, 588
diyOp, 604
exportExtra, 592
exportOp, 590, 591
genConv2dSep, 597
genConv3dSep, 598
genConvSeparated, 596
generalizedConvolution, 594
generalizedConvolutionK1d, 595
geometricOp2d, 604
HxImageData, 587
ident, 588
import, 589, 590
inout, 591
MNPixOp, 594
multiPixOp, 594
name, 588
neighbourhoodOp, 601, 603
neighbourhoodOpExtra, 601
neighbourhoodOpExtra2, 602
numberOfPixels, 589
pixelDimensionality, 589
pixelPrecision, 589
pixelType, 589
queueBasedOp, 603
recGenConv, 599
recGenConv2dSep, 600
rgbOp, 604
setBorder, 592
setPartImage, 592
signature, 589
sizes, 588
unaryPixOp, 593
HxImageFactory
getGenerator, 606
HxImageRep, 625
subscribeGenerator, 606
unSubscribeGenerator, 606
HxImageFactory, 605
from2Images, 611
from3Images, 611
fromByteData, 607
fromDoubleData, 609
fromFile, 612
fromFloatData, 608
fromGenerator, 609
fromGrayValue, 610
fromImage, 607

fromImport, 610
fromIntData, 608
fromJavaRgb, 610
fromMatlab, 610
fromNamedGenerator, 609
fromShortData, 608
fromSignature, 607
fromValue, 607
imagesFromFile, 615
imagesToFile, 614
instance, 607
writeFile, 613
HxImageList
begin, 617
const_iterator, 617
end, 617
HxImageList, 617, 618
iterator, 617
nImages, 617
operator+, 618
operator+=, 617
operator=, 617
operator[ ], 617
size, 617
HxImageList, 617
~HxImageList, 618
binaryPixOp, 619
HxImageList, 618
MNPixOp, 620
multiPixOp, 619
pointees, 618
unaryPixOp, 618
HxImageMaxSize
HxImageMaxSize.h, 256
HxImageMaxSize.h, 256
HxImageMaxSize, 256
HxImageMinSize
HxImageMinSize.h, 257
HxImageMinSize.h, 256
HxImageMinSize, 257
HxImageRep
dirty, 625
HxGetValues, 625
HxImageFactory, 625
HxImageRep, 626
HxImageRepInit, 625
ref, 625
setImageDataObserver, 625
setObjectObserver, 625
HxImageRep, 620
~HxImageRep, 626
binaryPixOp, 629, 630
dimensionality, 628
dimensionSize, 628
diyOp, 638
exportOp, 631
exportOpExtra, 631
genConv2dSep, 634
genConv3dSep, 634
genConvSeparated, 633
generalizedConvolution, 632
generalizedConvolutionK1d, 633
geometricOp2d, 637
getAt, 639
getResultPrecision, 632
getRgbPixels2d, 639, 640
getRgbPixels3d, 640
HxExportMatlabPixels, 646
HxExtend, 644
HxExtendVal, 645
HxImageRep, 626
HxInverseProjectRange, 641
HxProjectRange, 641
HxRestrict, 642
ident, 627
isNull, 627
MNPixOp, 630
multiPixOp, 630
name, 627
neighbourhoodOp, 636, 637
neighbourhoodOpExtra, 636
neighbourhoodOpExtra2, 636
numberOfPixels, 628
operator int, 627
operator=, 627
operator==, 627
pixelDimensionality, 628
pixelPrecision, 629
pixelType, 628
printInfo, 639
queueBasedOp, 637
recGenConv, 635
recGenConv2dSep, 635
reduceOp, 632
ResultPrecision, 626
setAt, 639
setDefaultResultPrecision, 632
signature, 629
sizes, 628
unaryPixOp, 629
HxImageRepInit
HxImageRep, 625
HxImageRepToMatrix
HxMatrixConv.h, 293
HxImageRepToVector
HxMatrixConv.h, 293
HxImageSeq
AVI_F, 648

HxImageSeq, 649
HxImageSeqIter, 648
MIR_F, 648
MPEG_F, 648
HxImageSeq, 646
~HxImageSeq, 649
begin, 652
end, 652
frameDepth, 651
frameHeight, 650
frameWidth, 650
getFrame, 651
getRgb2d, 651
getRgbPixels2d, 651
HxImageSeq, 649
ident, 650
isNull, 650
nrFrames, 651
operator=, 650
put, 653
setUseMDC, 650
writeFile, 652
HxImageSeqData
HxImageSeqData, 655
HxImageSeqData, 653
~HxImageSeqData, 655
frame2HxImageRep, 657
frameDepth, 656
frameHeight, 656
frameWidth, 655
getFrame, 656
getRgb2d, 656
getRgbPixels2d, 657
HxImageSeqData, 655
ident, 655
nrFrames, 656
valid, 655
HxImageSeqDXMedia
HxImageSeqDXMedia, 659
HxImageSeqDXMedia, 657
~HxImageSeqDXMedia, 659
frame2HxImageRep, 661
frameDepth, 660
frameHeight, 659
frameWidth, 659
getRgb2d, 660
getRgbPixels2d, 660
HxImageSeqDXMedia, 659
nrFrames, 660
valid, 659
HxImageSeqIter
HxImageSeq, 648
HxImageSeqIter, 662, 663
HxImageSeqIter, 661
~HxImageSeqIter, 663
clone, 664
HxImageSeqIter, 662, 663
operator ∗, 664
operator!=, 665
operator++, 663
operator+=, 664
operator–, 664
operator=, 663
operator==, 665
HxImageSeqMDC
HxImageSeqMDC, 666
HxImageSeqMDC, 665
~HxImageSeqMDC, 667
frame2HxImageRep, 669
frameDepth, 668
frameHeight, 667
frameWidth, 667
getRgb2d, 668
getRgbPixels2d, 668
HxImageSeqMDC, 666
nrFrames, 668
valid, 667
HxImagesFromFile
HxImagesFromFile.h, 257
HxImagesFromFile.h, 257
HxImagesFromFile, 257
HxImageSig2dByte
Allocator, 670
ArithImageSigType, 670
ArithImageSigTypeDouble, 670
ArithType, 670
ArithTypeDouble, 670
DataPtrType, 670
HxImageSig2dByte, 670
PixelType, 670
ProjectDomainImageSigType, 670
HxImageSig2dByte, 669
HxImageSig2dComplex
Allocator, 670
ArithImageSigType, 670
ArithImageSigTypeDouble, 670
ArithType, 670
ArithTypeDouble, 670
DataPtrType, 670
HxImageSig2dComplex, 671
PixelType, 670
ProjectDomainImageSigType, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble
Allocator, 671
ArithImageSigType, 671
ArithImageSigTypeDouble, 671
ArithType, 671

ArithTypeDouble, 671
DataPtrType, 671
HxImageSig2dDouble, 671
PixelType, 671
ProjectDomainImageSigType, 671
HxImageSig2dDouble, 671
HxImageSig2dFloat
Allocator, 672
ArithImageSigType, 672
ArithImageSigTypeDouble, 672
ArithType, 672
ArithTypeDouble, 672
DataPtrType, 672
HxImageSig2dFloat, 672
PixelType, 672
ProjectDomainImageSigType, 672
HxImageSig2dFloat, 672
HxImageSig2dInt
Allocator, 673
ArithImageSigType, 673
ArithImageSigTypeDouble, 673
ArithType, 673
ArithTypeDouble, 673
DataPtrType, 673
HxImageSig2dInt, 673
PixelType, 673
ProjectDomainImageSigType, 673
HxImageSig2dInt, 672
HxImageSig2dShort
Allocator, 673
ArithImageSigType, 673
ArithImageSigTypeDouble, 673
ArithType, 673
ArithTypeDouble, 673
DataPtrType, 673
HxImageSig2dShort, 674
PixelType, 673
ProjectDomainImageSigType, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte
Allocator, 674
ArithImageSigType, 674
ArithImageSigTypeDouble, 674
ArithType, 674
ArithTypeDouble, 674
DataPtrType, 674
HxImageSig2dVec2Byte, 674
PixelType, 674
ProjectDomainImageSigType, 674
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double
Allocator, 675
ArithImageSigType, 675
ArithImageSigTypeDouble, 675
ArithType, 675
ArithTypeDouble, 675
DataPtrType, 675
HxImageSig2dVec2Double, 675
PixelType, 675
ProjectDomainImageSigType, 675
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float
Allocator, 676
ArithImageSigType, 676
ArithImageSigTypeDouble, 676
ArithType, 676
ArithTypeDouble, 676
DataPtrType, 676
HxImageSig2dVec2Float, 676
PixelType, 676
ProjectDomainImageSigType, 676
HxImageSig2dVec2Float, 675
HxImageSig2dVec2Int
Allocator, 676
ArithImageSigType, 676
ArithImageSigTypeDouble, 676
ArithType, 676
ArithTypeDouble, 676
DataPtrType, 676
HxImageSig2dVec2Int, 677
PixelType, 676
ProjectDomainImageSigType, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short
Allocator, 677
ArithImageSigType, 677
ArithImageSigTypeDouble, 677
ArithType, 677
ArithTypeDouble, 677
DataPtrType, 677
HxImageSig2dVec2Short, 677
PixelType, 677
ProjectDomainImageSigType, 677
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte
Allocator, 678
ArithImageSigType, 678
ArithImageSigTypeDouble, 678
ArithType, 678
ArithTypeDouble, 678
DataPtrType, 678
HxImageSig2dVec3Byte, 678
PixelType, 678
ProjectDomainImageSigType, 678
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double
Allocator, 679
ArithImageSigType, 679

ArithImageSigTypeDouble, 679
ArithType, 679
ArithTypeDouble, 679
DataPtrType, 679
HxImageSig2dVec3Double, 679
PixelType, 679
ProjectDomainImageSigType, 679
HxImageSig2dVec3Double, 678
HxImageSig2dVec3Float
Allocator, 679
ArithImageSigType, 679
ArithImageSigTypeDouble, 679
ArithType, 679
ArithTypeDouble, 679
DataPtrType, 679
HxImageSig2dVec3Float, 680
PixelType, 679
ProjectDomainImageSigType, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int
Allocator, 680
ArithImageSigType, 680
ArithImageSigTypeDouble, 680
ArithType, 680
ArithTypeDouble, 680
DataPtrType, 680
HxImageSig2dVec3Int, 680
PixelType, 680
ProjectDomainImageSigType, 680
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short
Allocator, 681
ArithImageSigType, 681
ArithImageSigTypeDouble, 681
ArithType, 681
ArithTypeDouble, 681
DataPtrType, 681
HxImageSig2dVec3Short, 681
PixelType, 681
ProjectDomainImageSigType, 681
HxImageSig2dVec3Short, 681
HxImageSig3dByte
Allocator, 682
ArithImageSigType, 682
ArithImageSigTypeDouble, 682
ArithType, 682
ArithTypeDouble, 682
DataPtrType, 682
HxImageSig3dByte, 682
PixelType, 682
ProjectDomainImageSigType, 682
HxImageSig3dByte, 681
HxImageSig3dDouble
Allocator, 682
ArithImageSigType, 682
ArithImageSigTypeDouble, 682
ArithType, 682
ArithTypeDouble, 682
DataPtrType, 682
HxImageSig3dDouble, 683
PixelType, 682
ProjectDomainImageSigType, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat
Allocator, 683
ArithImageSigType, 683
ArithImageSigTypeDouble, 683
ArithType, 683
ArithTypeDouble, 683
DataPtrType, 683
HxImageSig3dFloat, 683
PixelType, 683
ProjectDomainImageSigType, 683
HxImageSig3dFloat, 683
HxImageSig3dInt
Allocator, 684
ArithImageSigType, 684
ArithImageSigTypeDouble, 684
ArithType, 684
ArithTypeDouble, 684
DataPtrType, 684
HxImageSig3dInt, 684
PixelType, 684
ProjectDomainImageSigType, 684
HxImageSig3dInt, 684
HxImageSig3dShort
Allocator, 685
ArithImageSigType, 685
ArithImageSigTypeDouble, 685
ArithType, 685
ArithTypeDouble, 685
DataPtrType, 685
HxImageSig3dShort, 685
PixelType, 685
ProjectDomainImageSigType, 685
HxImageSig3dShort, 684
HxImageSignature
_imageDimensionality, 688
_pixelDimensionality, 688
_pixelPrecision, 688
_pixelType, 688
HxImageSignature, 688
ident, 688
HxImageSignature, 685
broadest, 689
HxImageSignature, 688
imageDimensionality, 691
isEqual, 689

NameToSignature, 693
operator!=, 690
operator<, 690
operator=, 689
operator==, 690
pixelDimensionality, 691
pixelPrecision, 691
pixelType, 691
put, 692
setImageDimensionality, 691
setPixelDimensionality, 691
setPixelPrecision, 692
setPixelType, 692
toString, 692
HxImagesToFile
HxImagesToFile.h, 258
HxImagesToFile.h, 258
HxImagesToFile, 258
HxImageTem
_dimSizes, 695
~HxImageTem, 694
ArithImageSigType, 694
ArithImageSigTypeDouble, 694
ArithType, 694
ArithTypeDouble, 694
dataPtrClone, 695
DataPtrType, 694
getAt, 695
getDoublePixels, 695
getValues, 695
HxImageTem, 694
makeScratch, 695
printInfo, 695
set, 694
setAt, 695
HxImageTem, 693
dimensionality, 695
dimensionSize, 695
numberOfPixels, 696
pixelDimensionality, 696
pixelPrecision, 696
pixelType, 696
signature, 696
sizes, 695
HxImageTem2d
~HxImageTem2d, 697
dataPtrClone, 697
getRgbPixels2d, 698
getValues, 697
height, 697
HxImageTem2d, 697
inverseProjectDomain, 697
projectDomain, 697
width, 697
HxImageTem2d, 697
geometricOp2d, 698
HxImageTem3d
~HxImageTem3d, 699
dataPtrClone, 699
depth, 699
getValues, 699
height, 699
HxImageTem3d, 699
inverseProjectDomain, 699
projectDomain, 699
width, 699
HxImageTem3d, 699
geometricOp2d, 700
HxImageToSegmentation
HxImageToSegmentation.h, 258
HxImageToSegmentation.h, 258
HxImageToSegmentation, 258
HxImgFtorBpo
HxImgFtorBpo, 702
HxImgFtorBpo, 700
~HxImgFtorBpo, 702
doIt, 703
HxImgFtorBpo, 702
KeyType, 701
RuleType, 701
HxImgFtorBpoKey
HxImgFtorBpoKey, 704
HxImgFtorBpoKey, 703
HxImgFtorBpoKey, 704
HxImgFtorDescription
~HxImgFtorDescription, 704
addArgument, 704
HxImgFtorDescription, 704
operator<, 704
operator=, 704
operator==, 704
put, 704
setKey, 704
setTags, 704
setVariation, 704
toString, 704
HxImgFtorDescription, 704
HxImgFtorDiy
HxImgFtorDiy, 706
HxImgFtorDiy, 705
~HxImgFtorDiy, 706
doIt, 707
HxImgFtorDiy, 706
KeyType, 706
HxImgFtorDiyKey
HxImgFtorDiyKey, 708
HxImgFtorDiyKey, 707
HxImgFtorDiyKey, 708

HxImgFtorExportExtra
HxImgFtorExportExtra, 709
HxImgFtorExportExtra, 708
~HxImgFtorExportExtra, 709
doIt, 710
HxImgFtorExportExtra, 709
KeyType, 709
HxImgFtorExportExtraKey
HxImgFtorExportExtraKey, 711
HxImgFtorExportExtraKey, 710
HxImgFtorExportExtraKey, 711
HxImgFtorGenConv2d
HxImgFtorGenConv2d, 713
HxImgFtorGenConv2d, 711
~HxImgFtorGenConv2d, 713
doIt, 713
HxImgFtorGenConv2d, 713
KeyType, 713
HxImgFtorGenConv2dK1d
HxImgFtorGenConv2dK1d, 715
HxImgFtorGenConv2dK1d, 714
~HxImgFtorGenConv2dK1d, 716
doIt, 716
HxImgFtorGenConv2dK1d, 715
KeyType, 715
HxImgFtorGenConv2dSep
HxImgFtorGenConv2dSep, 718
HxImgFtorGenConv2dSep, 717
~HxImgFtorGenConv2dSep, 719
doIt, 719
HxImgFtorGenConv2dSep, 718
KeyType, 718
HxImgFtorGenConv2dSepKey
HxImgFtorGenConv2dSepKey, 720
HxImgFtorGenConv2dSepKey, 720
HxImgFtorGenConv2dSepKey, 720
HxImgFtorGenConv3d
HxImgFtorGenConv3d, 722
HxImgFtorGenConv3d, 721
~HxImgFtorGenConv3d, 722
doIt, 723
HxImgFtorGenConv3d, 722
KeyType, 722
HxImgFtorGenConv3dK1d
HxImgFtorGenConv3dK1d, 725
HxImgFtorGenConv3dK1d, 723
~HxImgFtorGenConv3dK1d, 725
doIt, 725
HxImgFtorGenConv3dK1d, 725
KeyType, 725
HxImgFtorGenConvK1dKey
HxImgFtorGenConvK1dKey, 727
HxImgFtorGenConvK1dKey, 726
HxImgFtorGenConvK1dKey, 727
HxImgFtorGenConvKey
HxImgFtorGenConvKey, 728
HxImgFtorGenConvKey, 727
HxImgFtorGenConvKey, 728
HxImgFtorI1
HxImgFtorI1, 730
HxImgFtorI1, 728
~HxImgFtorI1, 730
callIt, 731
HxImgFtorI1, 730
KeyType, 729
HxImgFtorI1Cast
HxImgFtorI1Cast, 733
HxImgFtorI1Cast, 731
~HxImgFtorI1Cast, 734
callIt, 734
doIt, 734
HxImgFtorI1Cast, 733
ImgDataPtrType, 733
KeyType, 732
HxImgFtorI1CastKey
HxImgFtorI1CastKey, 736
HxImgFtorI1CastKey, 736
HxImgFtorI1CastKey, 736
HxImgFtorI1Key
HxImgFtorI1Key, 737
HxImgFtorI1Key, 736
HxImgFtorI1Key, 737
HxImgFtorI2
HxImgFtorI2, 742
HxImgFtorI2, 737
~HxImgFtorI2, 742
callIt, 743
HxImgFtorI2, 742
KeyType, 738
HxImgFtorI2Cast
HxImgFtorI2Cast, 749
HxImgFtorI2Cast, 743
~HxImgFtorI2Cast, 749
callIt, 749
doIt, 750
HxImgFtorI2Cast, 749
Img1DataPtrType, 748
Img2DataPtrType, 749
KeyType, 745
HxImgFtorI2CastKey
HxImgFtorI2CastKey, 754
HxImgFtorI2CastKey, 754
HxImgFtorI2CastKey, 754
HxImgFtorI2Key
HxImgFtorI2Key, 755
HxImgFtorI2Key, 755
HxImgFtorI2Key, 755
HxImgFtorI3

HxImgFtorI3, 759
HxImgFtorI3, 756
~HxImgFtorI3, 759
callIt, 759
HxImgFtorI3, 759
KeyType, 756
HxImgFtorI3Cast
HxImgFtorI3Cast, 763
HxImgFtorI3Cast, 759
~HxImgFtorI3Cast, 763
callIt, 764
doIt, 764
HxImgFtorI3Cast, 763
Img1DataPtrType, 763
Img2DataPtrType, 763
Img3DataPtrType, 763
KeyType, 761
HxImgFtorI3CastKey
HxImgFtorI3CastKey, 768
HxImgFtorI3CastKey, 767
HxImgFtorI3CastKey, 768
HxImgFtorI3Key
HxImgFtorI3Key, 769
HxImgFtorI3Key, 768
HxImgFtorI3Key, 769
HxImgFtorI4
HxImgFtorI4, 771
HxImgFtorI4, 770
~HxImgFtorI4, 771
callIt, 771
HxImgFtorI4, 771
KeyType, 770
HxImgFtorI4Cast
HxImgFtorI4Cast, 774
HxImgFtorI4Cast, 772
~HxImgFtorI4Cast, 774
callIt, 775
doIt, 775
HxImgFtorI4Cast, 774
Img1DataPtrType, 774
Img2DataPtrType, 774
Img3DataPtrType, 774
Img4DataPtrType, 774
KeyType, 773
HxImgFtorI4CastKey
HxImgFtorI4CastKey, 777
HxImgFtorI4CastKey, 776
HxImgFtorI4CastKey, 777
HxImgFtorI4Key
HxImgFtorI4Key, 777
HxImgFtorI4Key, 777
HxImgFtorI4Key, 777
HxImgFtorIM
HxImgFtorIM, 779
HxImgFtorIM, 778
~HxImgFtorIM, 779
callIt, 779
HxImgFtorIM, 779
KeyType, 779
HxImgFtorIMCast
HxImgFtorIMCast, 781
HxImgFtorIMCast, 779
~HxImgFtorIMCast, 781
callIt, 782
doIt, 782
DstDataPtrType, 781
HxImgFtorIMCast, 781
KeyType, 781
SrcDataPtrArray, 781
SrcDataPtrType, 781
HxImgFtorIMCastKey
HxImgFtorIMCastKey, 783
HxImgFtorIMCastKey, 783
HxImgFtorIMCastKey, 783
HxImgFtorIMKey
HxImgFtorIMKey, 784
HxImgFtorIMKey, 784
HxImgFtorIMKey, 784
HxImgFtorIMN
HxImgFtorIMN, 786
HxImgFtorIMN, 784
~HxImgFtorIMN, 786
callIt, 786
HxImgFtorIMN, 786
KeyType, 785
HxImgFtorIMNCast
HxImgFtorIMNCast, 788
HxImgFtorIMNCast, 786
~HxImgFtorIMNCast, 788
callIt, 789
doIt, 789
DstDataPtrArray, 788
DstDataPtrType, 788
HxImgFtorIMNCast, 788
KeyType, 788
SrcDataPtrArray, 788
SrcDataPtrType, 788
HxImgFtorIMNCastKey
HxImgFtorIMNCastKey, 790
HxImgFtorIMNCastKey, 789
HxImgFtorIMNCastKey, 790
HxImgFtorIMNKey
HxImgFtorIMNKey, 791
HxImgFtorIMNKey, 790
HxImgFtorIMNKey, 791
HxImgFtorInOut
HxImgFtorInOut, 793
HxImgFtorInOut, 791

~HxImgFtorInOut, 793
doIt, 793
HxImgFtorInOut, 793
KeyType, 793
HxImgFtorInOutKey
HxImgFtorInOutKey, 795
HxImgFtorInOutKey, 794
HxImgFtorInOutKey, 795
HxImgFtorKernelNgb2d
HxImgFtorKernelNgb2d, 796
HxImgFtorKernelNgb2d, 795
~HxImgFtorKernelNgb2d, 796
doIt, 797
HxImgFtorKernelNgb2d, 796
KeyType, 796
probeOp, 797
HxImgFtorKernelNgbKey
HxImgFtorKernelNgbKey, 799
HxImgFtorKernelNgbKey, 798
HxImgFtorKernelNgbKey, 799
HxImgFtorKey
HxImgFtorKey, 800, 801
HxImgFtorKey, 799
addArgument, 801
compare, 802
getArgument, 801
getClassName, 802
HxImgFtorKey, 800, 801
put, 802
setArguments, 801
toString, 802
HxImgFtorKeyNameTable
~HxImgFtorKeyNameTable, 803
getClassName, 803
getClassNameId, 803
getName, 803
getNameId, 803
getTypeName, 803
getTypeNameId, 803
instance, 803
put, 803
SizeType, 803
HxImgFtorKeyNameTable, 803
HxImgFtorMNpo
HxImgFtorMNpo, 805
HxImgFtorMNpo, 804
~HxImgFtorMNpo, 805
doIt, 806
HxImgFtorMNpo, 805
KeyType, 805
probeOp, 805
HxImgFtorMNpoKey
HxImgFtorMNpoKey, 807
HxImgFtorMNpoKey, 806
HxImgFtorMNpoKey, 807
HxImgFtorMpo
HxImgFtorMpo, 808
HxImgFtorMpo, 807
~HxImgFtorMpo, 809
doIt, 809
HxImgFtorMpo, 808
KeyType, 808
HxImgFtorMpoKey
HxImgFtorMpoKey, 810
HxImgFtorMpoKey, 809
HxImgFtorMpoKey, 810
HxImgFtorNgb2d
HxImgFtorNgb2d, 812
HxImgFtorNgb2d, 810
~HxImgFtorNgb2d, 812
doIt, 812
HxImgFtorNgb2d, 812
KeyType, 812
probeOp, 812
HxImgFtorNgb2dExtra
HxImgFtorNgb2dExtra, 815
HxImgFtorNgb2dExtra, 813
~HxImgFtorNgb2dExtra, 815
doIt, 815
HxImgFtorNgb2dExtra, 815
KeyType, 814
probeOp, 815
HxImgFtorNgb2dExtra2
HxImgFtorNgb2dExtra2, 818
HxImgFtorNgb2dExtra2, 816
~HxImgFtorNgb2dExtra2, 818
doIt, 818
HxImgFtorNgb2dExtra2, 818
KeyType, 817
probeOp, 818
HxImgFtorNgbExtra2Key
HxImgFtorNgbExtra2Key, 820
HxImgFtorNgbExtra2Key, 819
HxImgFtorNgbExtra2Key, 820
HxImgFtorNgbExtraKey
HxImgFtorNgbExtraKey, 821
HxImgFtorNgbExtraKey, 820
HxImgFtorNgbExtraKey, 821
HxImgFtorNgbKey
HxImgFtorNgbKey, 822
HxImgFtorNgbKey, 821
HxImgFtorNgbKey, 822
HxImgFtorObserver, 822
inserted, 822
HxImgFtorQueueBased
~HxImgFtorQueueBased, 823
className, 823
createNeighborCoordinates, 823

fillNeighborValues, 823
HxImgFtorQueueBased, 823
QueueT, 823
queuet, 823
VecNeighborT, 823
VecPointT, 823
HxImgFtorQueueBased, 822
doIt, 824
HxImgFtorQueueBasedKey
HxImgFtorQueueBasedKey, 827
HxImgFtorQueueBasedKey, 827
HxImgFtorRecGenConv2d
HxImgFtorRecGenConv2d, 829
HxImgFtorRecGenConv2d, 828
~HxImgFtorRecGenConv2d, 829
doIt, 830
HxImgFtorRecGenConv2d, 829
KeyType, 829
HxImgFtorRecGenConv2dK1d
HxImgFtorRecGenConv2dK1d, 832
HxImgFtorRecGenConv2dK1d, 830
~HxImgFtorRecGenConv2dK1d, 832
doIt, 832
HxImgFtorRecGenConv2dK1d, 832
KeyType, 831
HxImgFtorRecGenConvK1dKey
HxImgFtorRecGenConvK1dKey, 834
HxImgFtorRecGenConvK1dKey, 833
HxImgFtorRecGenConvK1dKey, 834
HxImgFtorRecGenConvKey
HxImgFtorRecGenConvKey, 835
HxImgFtorRecGenConvKey, 834
HxImgFtorRecGenConvKey, 835
HxImgFtorRgb2d
HxImgFtorRgb2d, 836
HxImgFtorRgb2d, 835
~HxImgFtorRgb2d, 836
doIt, 837
HxImgFtorRgb2d, 836
KeyType, 836
HxImgFtorRgb3d
HxImgFtorRgb3d, 838
HxImgFtorRgb3d, 837
~HxImgFtorRgb3d, 839
doIt, 839
HxImgFtorRgb3d, 838
KeyType, 838
HxImgFtorRgbKey
HxImgFtorRgbKey, 840
HxImgFtorRgbKey, 840
HxImgFtorRgbKey, 840
HxImgFtorRuleBase, 840
~HxImgFtorRuleBase, 842
getArgumentType, 844
getExtra2Type, 846
getExtraType, 845
getIsModifying, 846
getKernelType, 845
getResultType, 843
instance, 843
QueryResultType, 842
setArgumentType, 844
setExtra2Type, 845
setExtraType, 845
setIsModifying, 846
setKernelType, 844
setResultType, 843
HxImgFtorSet
HxImgFtorSet, 848
HxImgFtorSet, 847
~HxImgFtorSet, 848
doIt, 848
HxImgFtorSet, 848
KeyType, 848
HxImgFtorSetBorder2d
HxImgFtorSetBorder2d, 851
HxImgFtorSetBorder2d, 849
~HxImgFtorSetBorder2d, 851
doIt, 851
HxImgFtorSetBorder2d, 851
KeyType, 850
HxImgFtorSetBorder3d
HxImgFtorSetBorder3d, 853
HxImgFtorSetBorder3d, 852
~HxImgFtorSetBorder3d, 853
doIt, 853
HxImgFtorSetBorder3d, 853
KeyType, 853
HxImgFtorSetBorderKey
HxImgFtorSetBorderKey, 855
HxImgFtorSetBorderKey, 854
HxImgFtorSetBorderKey, 855
HxImgFtorSetKey
HxImgFtorSetKey, 855
HxImgFtorSetKey, 855
HxImgFtorSetKey, 855
HxImgFtorTable
HxImgFtorTable, 856
HxImgFtorTable, 856
~HxImgFtorTable, 857
addImgFtorObserver, 858
find, 857
insert, 857
instance, 857
put, 858
HxImgFtorTableTem, 858
find, 859
HxImgFtorUpo

HxImgFtorUpo, 860
HxImgFtorUpo, 859
~HxImgFtorUpo, 860
doIt, 861
HxImgFtorUpo, 860
KeyType, 860
HxImgFtorUpoKey
HxImgFtorUpoKey, 862
HxImgFtorUpoKey, 861
HxImgFtorUpoKey, 862
HxImgFunctor
getDescription, 863
HxImgFunctor, 863
HxImgFunctor, 862
~HxImgFunctor, 863
HxImgFunctor, 863
probeOp, 863
put, 863
HxInf
HxInf.h, 259
HxInf.h, 259
HxInf, 259
HxInfimumReconstruction
HxInfimumReconstruction.h, 261
HxInfimumReconstruction.h, 260
HxInfimumReconstruction, 261
HxInfVal
HxInfVal.h, 261
HxInfVal.h, 261
HxInfVal, 261
HxInstantiatorAbs, 864
f, 864
HxInstantiatorAcos, 864
f, 865
HxInstantiatorAdd, 865
f, 865
HxInstantiatorAddReduce, 866
f, 866
HxInstantiatorAddSat, 866
f, 867
HxInstantiatorAddV, 867
f, 867
HxInstantiatorAnd, 868
f, 868
HxInstantiatorAndV, 868
f, 869
HxInstantiatorArg, 869
f, 869
HxInstantiatorAsin, 869
f, 870
HxInstantiatorAtan, 870
f, 870
HxInstantiatorAtan2, 871
f, 871
HxInstantiatorCeil, 871
f, 872
HxInstantiatorColSpace, 872
f, 872
HxInstantiatorComplement, 872
f, 873
HxInstantiatorConjugate, 873
f, 873
HxInstantiatorCos, 874
f, 874
HxInstantiatorCosh, 874
f, 875
HxInstantiatorCross, 875
f, 875
HxInstantiatorCrossV, 875
f, 876
HxInstantiatorDiv, 876
f, 876
HxInstantiatorDivV, 877
f, 877
HxInstantiatorDot, 877
f, 878
HxInstantiatorDotV, 878
f, 878
HxInstantiatorEqual, 879
f, 879
HxInstantiatorEqualV, 879
f, 880
HxInstantiatorExp, 880
f, 880
HxInstantiatorExpPix, 880
f, 881
HxInstantiatorFloor, 881
f, 881
HxInstantiatorGpi, 882
f, 882
HxInstantiatorGreaterEqual, 882
f, 883
HxInstantiatorGreaterEqualV, 883
f, 883
HxInstantiatorGreaterThan, 883
f, 884
HxInstantiatorGreaterThanV, 884
f, 885
HxInstantiatorHighlightRegion, 885
f, 885
HxInstantiatorImpPix, 885
f, 886
HxInstantiatorInf, 886
f, 886
HxInstantiatorInfReduce, 887
f, 887
HxInstantiatorInfV, 887
f, 888

HxInstantiatorLeftShift, 888
f, 888
HxInstantiatorLeftShiftV, 889
f, 889
HxInstantiatorLessEqual, 889
f, 890
HxInstantiatorLessEqualV, 890
f, 890
HxInstantiatorLessThan, 890
f, 891
HxInstantiatorLessThanV, 891
f, 892
HxInstantiatorLog, 892
f, 892
HxInstantiatorLog10, 892
f, 893
HxInstantiatorMax, 893
f, 893
HxInstantiatorMaxReduce, 894
f, 894
HxInstantiatorMaxV, 894
f, 895
HxInstantiatorMin, 895
f, 895
HxInstantiatorMinReduce, 895
f, 896
HxInstantiatorMinV, 896
f, 897
HxInstantiatorMod, 897
f, 897
HxInstantiatorModV, 897
f, 898
HxInstantiatorMul, 898
f, 899
HxInstantiatorMulReduce, 899
f, 899
HxInstantiatorMulV, 899
f, 900
HxInstantiatorNegate, 900
f, 901
HxInstantiatorNorm1, 901
f, 901
HxInstantiatorNorm2, 901
f, 902
HxInstantiatorNorm2Sqr, 902
f, 902
HxInstantiatorNormInf, 903
f, 903
HxInstantiatorNotEqual, 903
f, 904
HxInstantiatorNotEqualV, 904
f, 904
HxInstantiatorOr, 904
f, 905
HxInstantiatorOrV, 905
f, 906
HxInstantiatorPow, 906
f, 906
HxInstantiatorPowV, 906
f, 907
HxInstantiatorProduct, 907
f, 908
HxInstantiatorRGB2Intensity, 908
f, 908
HxInstantiatorRightShift, 908
f, 909
HxInstantiatorRightShiftV, 909
f, 909
HxInstantiatorRound, 910
f, 910
HxInstantiatorSet, 910
f001, 912
f002, 912
f003, 912
f004, 912
f005, 912
f006, 912
f007, 912
f008, 912
f009, 913
f010, 913
f011, 913
f012, 913
f013, 913
f014, 913
f015, 913
f016, 913
f017, 913
f018, 914
f019, 914
f020, 914
f021, 914
HxInstantiatorSetPartImg, 914
f, 915
HxInstantiatorSetVal, 915
f, 915
HxInstantiatorSin, 915
f, 916
HxInstantiatorSinh, 916
f, 916
HxInstantiatorSpi, 916
f, 917
HxInstantiatorSqrDst, 917
f, 917
HxInstantiatorSqrt, 918
f, 918
HxInstantiatorSub, 918
f, 919

HxInstantiatorSubSat, 919
f, 919
HxInstantiatorSubV, 920
f, 920
HxInstantiatorSum, 920
f, 921
HxInstantiatorSup, 921
f, 921
HxInstantiatorSupReduce, 921
f, 922
HxInstantiatorSupV, 922
f, 922
HxInstantiatorTan, 923
f, 923
HxInstantiatorTanh, 923
f, 924
HxInstantiatorTriStateThreshold, 924
f, 924
HxInstantiatorUpoMax, 924
f, 925
HxInstantiatorUpoMin, 925
f, 925
HxInstantiatorUpoThreshold, 926
f, 926
HxInstantiatorVec2, 926
f, 927
HxInstantiatorVec3, 927
f, 927
HxInstantiatorXor, 927
f, 928
HxInstantiatorXorV, 928
f, 928
HxInstDiyTranspose, 929
f, 929
HxInstExportExtraIdentMaskCentralMoments, 929
f, 930
HxInstExportExtraIdentMaskMean, 930
f, 930
HxInstExportExtraIdentMaskMedian, 931
f, 931
HxInstExportExtraIdentMaskMoments, 931
f, 932
HxInstExportExtraIdentMaskStdev, 932
f, 933
HxInstExportExtraIdentMaskSum, 933
f, 933
HxInstExportExtraWeightMaskSum, 934
f, 934
HxInstExpPpm, 934
f, 935
HxInstGenConv2dAddInf, 935
f, 935
HxInstGenConv2dAddMax, 936
f, 936
HxInstGenConv2dAddMin, 936
f, 937
HxInstGenConv2dAddSup, 937
f, 938
HxInstGenConv2dK1dAddInf, 938
f, 938
HxInstGenConv2dK1dAddMax, 939
f, 939
HxInstGenConv2dK1dAddMin, 939
f, 940
HxInstGenConv2dK1dAddSup, 940
f, 941
HxInstGenConv2dK1dMulAdd, 941
f, 941
HxInstGenConv2dMulAdd, 942
f, 942
HxInstGenConv2dSepAddInf, 942
f, 943
HxInstGenConv2dSepAddMax, 943
f, 944
HxInstGenConv2dSepAddMin, 944
f, 944
HxInstGenConv2dSepAddSup, 945
f, 945
HxInstGenConv2dSepMulAdd, 945
f, 946
HxInstGenConv3dK1dMulAdd, 946
f, 947
HxInstGenConv3dMulAdd, 947
f, 947
HxInstGeneratePix, 948
f, 948
HxInstImpBytes, 948
f, 949
HxInstImpPackRgb, 949
f, 949
HxInstImpPpm, 949
f, 950
HxInstInOutGetPoints, 950
f, 950
HxInstKerNgb2dNormCorrelation, 950
f, 951
HxInstNgb2dMean, 951
f, 952
HxInstNgbIsMaxGradDir2d, 952
f, 952
HxInstNgbLWshed2d, 952
f, 953
HxInstNgbNonMaxSuppression2d, 953
f, 953
HxInstNgbPercentile2d, 954
f, 954
HxInstRecGenConv2dAddMin, 954

f, 955
HxInstRecGenConv2dK1dAddMin, 955
f, 955
HxInstRecGenConv2dK1dMulAdd, 956
f, 956
HxInstRecGenConv2dMulAdd, 956
f, 957
HxInstRgb2dBinary, 957
f, 957
HxInstRgb2dCMY, 958
f, 958
HxInstRgb2dDirect, 958
f, 959
HxInstRgb2dDirectNC, 959
f, 959
HxInstRgb2dHSI, 959
f, 960
HxInstRgb2dLab, 960
f, 960
HxInstRgb2dLabel, 961
f, 961
HxInstRgb2dLogMag, 961
f, 962
HxInstRgb2dLuv, 962
f, 962
HxInstRgb2dOOO, 962
f, 963
HxInstRgb2dStretch, 963
f, 963
HxInstRgb2dXYZ, 964
f, 964
HxInstRgb3dBinary, 964
f, 965
HxInstRgb3dCMY, 965
f, 965
HxInstRgb3dDirect, 965
f, 966
HxInstRgb3dHSI, 966
f, 966
HxInstRgb3dLab, 967
f, 967
HxInstRgb3dLabel, 967
f, 968
HxInstRgb3dLogMag, 968
f, 968
HxInstRgb3dLuv, 968
f, 969
HxInstRgb3dOOO, 969
f, 969
HxInstRgb3dStretch, 970
f, 970
HxInstRgb3dXYZ, 970
f, 971
HxInverseProjectRange
HxImageRep, 641
HxInverseProjectRange.h, 262
HxInverseProjectRange.h, 262
HxInverseProjectRange, 262
HxKernel1d
~HxKernel1d, 971
ArithType, 971
className, 971
HxKernel1d, 971
operator(), 971
sizes, 971
HxKernel1d, 971
HxKernel2d
~HxKernel2d, 972
ArithType, 972
className, 972
HxKernel2d, 972
operator(), 972
sizes, 972
HxKernel2d, 971
HxKernel3d
~HxKernel3d, 972
ArithType, 972
className, 972
HxKernel3d, 972
operator(), 972
sizes, 972
HxKernel3d, 972
HxKerNgbNormCorrelation
HxKerNgbNormCorrelation, 974
HxKerNgbNormCorrelation, 973
~HxKerNgbNormCorrelation, 974
className, 976
HxKerNgbNormCorrelation, 974
init, 975
init2, 975
IteratorCategory, 974
next, 975
next2, 975
PhaseCategory, 974
result, 976
size, 975
HxKuwahara
HxKuwahara.h, 264
HxKuwahara.h, 263
HxKuwahara, 264
HxLabel
HxLabel.h, 264
HxLabel.h, 264
HxLabel, 264
HxLabel2
HxLabel2.h, 265
HxLabel2.h, 265
HxLabel2, 265

HxLeftShift
HxLeftShift.h, 265
HxLeftShift.h, 265
HxLeftShift, 265
HxLeftShiftVal
HxLeftShiftVal.h, 267
HxLeftShiftVal.h, 267
HxLeftShiftVal, 267
HxLessEqual
HxLessEqual.h, 268
HxLessEqual.h, 268
HxLessEqual, 268
HxLessEqualVal
HxLessEqualVal.h, 269
HxLessEqualVal.h, 269
HxLessEqualVal, 269
HxLessThan
HxLessThan.h, 270
HxLessThan.h, 270
HxLessThan, 270
HxLessThanVal
HxLessThanVal.h, 271
HxLessThanVal.h, 271
HxLessThanVal, 271
HxLocalInterpol
dump, 977
HxLocalInterpol, 977
HxLocalInterpol, 976
~HxLocalInterpol, 978
allKnots, 978
allP, 978
HxLocalInterpol, 977
numKnots, 978
numP, 978
HxLocalMode
HxLocalMode.h, 272
HxLocalMode.h, 272
HxLocalMode, 272
HxLog
HxLog.h, 273
HxLog.h, 272
HxLog, 273
HxLog10
HxLog10.h, 274
HxLog10.h, 273
HxLog10, 274
HxLUT
HxLUT.h, 275
HxLUT.h, 274
HxLUT, 275
HxMakeFrom2Images
HxMakeFrom2Images.h, 275
HxMakeFrom2Images.h, 275
HxMakeFrom2Images, 275
HxMakeFrom3Images
HxMakeFrom3Images.h, 277
HxMakeFrom3Images.h, 276
HxMakeFrom3Images, 277
HxMakeFromByteData
HxMakeFromByteData.h, 278
HxMakeFromByteData.h, 278
HxMakeFromByteData, 278
HxMakeFromDoubleData
HxMakeFromDoubleData.h, 279
HxMakeFromDoubleData.h, 279
HxMakeFromDoubleData, 279
HxMakeFromFile
HxMakeFromFile.h, 280
HxMakeFromFile.h, 280
HxMakeFromFile, 280
HxMakeFromFloatData
HxMakeFromFloatData.h, 281
HxMakeFromFloatData.h, 280
HxMakeFromFloatData, 281
HxMakeFromGenerator
HxMakeFromGenerator.h, 282
HxMakeFromGenerator.h, 281
HxMakeFromGenerator, 282
HxMakeFromGrayValue
HxMakeFromGrayValue.h, 282
HxMakeFromGrayValue.h, 282
HxMakeFromGrayValue, 282
HxMakeFromImage
HxMakeFromImage.h, 283
HxMakeFromImage.h, 283
HxMakeFromImage, 283
HxMakeFromImport
HxMakeFromImport.h, 284
HxMakeFromImport.h, 284
HxMakeFromImport, 284
HxMakeFromIntData
HxMakeFromIntData.h, 284
HxMakeFromIntData.h, 284
HxMakeFromIntData, 284
HxMakeFromJavaRgb
HxMakeFromJavaRgb.h, 285
HxMakeFromJavaRgb.h, 285
HxMakeFromJavaRgb, 285
HxMakeFromMatlab
HxMakeFromMatlab.h, 287
HxMakeFromMatlab.h, 286
HxMakeFromMatlab, 287
HxMakeFromNamedGenerator
HxMakeFromNamedGenerator.h, 288
HxMakeFromNamedGenerator.h, 287
HxMakeFromNamedGenerator, 288
HxMakeFromPpmPixels
HxMakeFromPpmPixels.h, 288

HxMakeFromPpmPixels.h
HxMakeFromPpmPixels, 288
HxMakeFromPpmPixels.h, 288
HxMakeFromShortData
HxMakeFromShortData.h, 289
HxMakeFromShortData.h, 288
HxMakeFromShortData, 289
HxMakeFromSignature
HxMakeFromSignature.h, 289
HxMakeFromSignature.h, 289
HxMakeFromSignature, 289
HxMakeFromValue
HxMakeFromValue.h, 290
HxMakeFromValue.h, 290
HxMakeFromValue, 290
HxMakeGaussian1d
HxMakeGaussian1d.h, 292
HxMakeGaussian1d.h, 291
HxMakeGaussian1d, 292
HxMakeParabola1d
HxMakeParabola1d.h, 293
HxMakeParabola1d.h, 292
HxMakeParabola1d, 293
HxMakeTagList
HxTagList.h, 366
HxMatrix
∼HxMatrix, 984
HxMatrix, 984–988
HxVector, 984, 1345
put, 984
HxMatrix, 979
abs, 1000
add, 997
camera, 993
cos, 999
cosh, 999
div, 998
exp, 1000
HxMatrix, 984–988
i, 995
lift2dTo3dXY, 993
log, 1000
map, 1000
mul, 998
nCol, 994
nElem, 994
nRow, 994
operator ∗, 1001, 1002, 1005
operator!=, 1005
operator+, 1003
operator-, 995, 1004
operator/, 1002, 1003
operator=, 994
operator==, 1005
operator[], 995
projection, 993
reflect2d, 990
reflect3d, 992
rotate2d, 989
rotate2dDeg, 989
rotateX3d, 991
rotateX3dDeg, 991
rotateY3d, 991
rotateY3dDeg, 992
rotateZ3d, 992
rotateZ3dDeg, 992
scale2d, 989
scale3d, 990
sgn, 1000
shear2d, 990
sin, 999
sinh, 999
sqrt, 1000
sub, 997, 998
svd, 996
t, 996
tan, 999
tanh, 999
translate2d, 989
translate3d, 990
valid, 994
HxMatrixConv.h
HxImageRepToMatrix, 293
HxImageRepToVector, 293
HxMatrixToImageRep, 293
HxVectorToImageRep, 293
HxMatrixConv.h, 293
HxMatrixToImageRep
HxMatrixConv.h, 293
HxMax
HxMax.h, 294
HxMax.h, 293
HxMax, 294
HxMaxVal
HxMaxVal.h, 295
HxMaxVal.h, 295
HxMaxVal, 295
HxMfBpo
HxMfBpo, 1007
HxMfBpo, 1006
∼HxMfBpo, 1008
HxMfBpo, 1007
preOpIsOk, 1009
result, 1009
source1, 1008
source2, 1009
HxMfDiy
HxMfDiy, 1010

HxMfDiy, 1009
~HxMfDiy, 1010
HxMfDiy, 1010
result, 1010
source, 1010
HxMfExportExtra
HxMfExportExtra, 1012
HxMfExportExtra, 1011
~HxMfExportExtra, 1012
extra, 1012
HxMfExportExtra, 1012
preOpIsOk, 1012
source, 1012
HxMfGenConv
HxMfGenConv, 1014, 1015
HxMfGenConv, 1013
~HxMfGenConv, 1015
HxMfGenConv, 1014, 1015
kernel, 1015
kernel2, 1015
kernel3, 1016
preOpIsOk, 1016
result, 1016
source, 1015
HxMfIdentity
HxMfIdentity, 1017
HxMfIdentity, 1016
~HxMfIdentity, 1017
HxMfIdentity, 1017
result, 1018
source, 1017
HxMfKernelNgb
HxMfKernelNgb, 1019
HxMfKernelNgb, 1018
~HxMfKernelNgb, 1019
HxMfKernelNgb, 1019
kernel, 1020
preOpIsOk, 1020
result, 1020
source, 1020
HxMfMNpo
HxMfMNpo, 1021
HxMfMNpo, 1020
~HxMfMNpo, 1022
HxMfMNpo, 1021
preOpIsOk, 1023
resultCnt, 1023
results, 1023
sourceCnt, 1023
sources, 1023
HxMfMpo
HxMfMpo, 1024
HxMfMpo, 1024
~HxMfMpo, 1025
HxMfMpo, 1024
nSources, 1025
result, 1026
sources, 1025
HxMfNgb
HxMfNgb, 1027
HxMfNgb, 1026
~HxMfNgb, 1028
extra, 1028
extra2, 1028
HxMfNgb, 1027
preOpIsOk, 1028
result, 1028
source, 1028
HxMfQueueBased
HxMfQueueBased, 1030
HxMfQueueBased, 1029
~HxMfQueueBased, 1030
HxMfQueueBased, 1030
kernel, 1031
preOpIsOk, 1031
result, 1031
source, 1031
HxMfResize
HxMfResize, 1032
HxMfResize, 1031
~HxMfResize, 1032
argument, 1033
HxMfResize, 1032
result, 1033
source, 1033
HxMfUpo
HxMfUpo, 1034
HxMfUpo, 1033
~HxMfUpo, 1034
HxMfUpo, 1034
result, 1035
source, 1035
HxMin
HxMin.h, 296
HxMin.h, 296
HxMin, 296
HxMinVal
HxMinVal.h, 297
HxMinVal.h, 297
HxMinVal, 297
HxMod
HxMod.h, 298
HxMod.h, 298
HxMod, 298
HxModVal
HxModVal.h, 299
HxModVal.h, 299
HxModVal, 299

HxMorphologicalContour
HxMorphologicalContour.h, 300
HxMorphologicalContour.h
HxMorphologicalContour, 300
HxMorphologicalContour.h, 300
HxMorphologicalGradient
HxMorphologicalGradient.h, 301
HxMorphologicalGradient.h, 300
HxMorphologicalGradient, 301
HxMorphologicalGradient2
HxMorphologicalGradient2.h, 301
HxMorphologicalGradient2.h, 301
HxMorphologicalGradient2, 301
HxMul
HxMul.h, 302
HxMul.h, 301
HxMul, 302
HxMulVal
HxMulVal.h, 303
HxMulVal.h, 303
HxMulVal, 303
HxNameTable
HxNameTable, 1036
HxNameTable, 1035
~HxNameTable, 1036
getId, 1037
getName, 1037
getNames, 1037
HxNameTable, 1036
insert, 1037
put, 1038
sizeType, 1036
HxNegate
HxNegate.h, 304
HxNegate.h, 304
HxNegate, 304
HxNgbBernsen
~HxNgbBernsen, 1039
HxNgbBernsen, 1039
init, 1039
next, 1039
result, 1039
size, 1039
HxNgbBernsen, 1038
className, 1040
HxNgbBernsen, 1039
IteratorCategory, 1039
PhaseCategory, 1039
HxNgbDefuz
~HxNgbDefuz, 1040
HxNgbDefuz, 1041
init, 1040
next, 1040
result, 1040
size, 1040
HxNgbDefuz, 1040
className, 1041
HxNgbDefuz, 1041
IteratorCategory, 1041
PhaseCategory, 1041
HxNgbHilditch
~HxNgbHilditch, 1042
HxNgbHilditch, 1043
init, 1042
next, 1042
result, 1042
size, 1042
HxNgbHilditch, 1042
className, 1043
HxNgbHilditch, 1043
IteratorCategory, 1043
PhaseCategory, 1043
HxNgbIsMaxGradDir2d
HxNgbIsMaxGradDir2d, 1045
HxNgbIsMaxGradDir2d, 1044
~HxNgbIsMaxGradDir2d, 1045
begin, 1046
className, 1047
CnumType, 1045
end, 1046
HxNgbIsMaxGradDir2d, 1045
init, 1046
IteratorCategory, 1045
next, 1047
PhaseCategory, 1045
result, 1047
size, 1046
HxNgbKuwahara
~HxNgbKuwahara, 1048
HxNgbKuwahara, 1049
init, 1048
next, 1048
result, 1048
size, 1048
HxNgbKuwahara, 1048
className, 1049
HxNgbKuwahara, 1049
IteratorCategory, 1048
PhaseCategory, 1048
HxNgbLocalMode
check, 1050
HxNgbLocalMode, 1050
HxNgbLocalMode, 1049
className, 1052
HxNgbLocalMode, 1050
init, 1051
IteratorCategory, 1050
next, 1051

PhaseCategory, 1050
result, 1051
size, 1051
HxNgbLocalModeInst, 1052
f, 1052
HxNgbLWshed2d
HxNgbLWshed2d, 1054
HxNgbLWshed2d, 1053
~HxNgbLWshed2d, 1054
className, 1056
HxNgbLWshed2d, 1054
init, 1054
IteratorCategory, 1054
next, 1055
PhaseCategory, 1054
result, 1055
size, 1054
HxNgbNonMaxSuppression2d
HxNgbNonMaxSuppression2d, 1058
HxNgbNonMaxSuppression2d, 1056
~HxNgbNonMaxSuppression2d, 1058
begin, 1058
className, 1059
CnumType, 1057
end, 1058
HxNgbNonMaxSuppression2d, 1058
init, 1059
IteratorCategory, 1057
next, 1059
PhaseCategory, 1057
result, 1059
size, 1058
HxNgbOpticalFlowInst, 1060
f, 1060
HxNgbPercentile2d
HxNgbPercentile2d, 1062
HxNgbPercentile2d, 1060
~HxNgbPercentile2d, 1062
className, 1063
HxNgbPercentile2d, 1062
init, 1062
IteratorCategory, 1061
next, 1062
PhaseCategory, 1061
result, 1063
size, 1062
HxNJet
HxNJet, 1066
normalize, 1065
resample, 1065
rotate, 1065
rotateDeg, 1065
truncate, 1065
HxNJet, 1063
~HxNJet, 1067
getJidx, 1069
getJList, 1070
getJw, 1071
getLidx, 1069
getList, 1070
getLList, 1070
getLw, 1071
getMidx, 1069
getMList, 1070
getMw, 1071
HxNJet, 1066
ident, 1067
isColor, 1068
nrComponents, 1068
operator=, 1067
ord2idx, 1071, 1072
order, 1067
put, 1071
scale, 1068
toFile, 1067
xy, 1068
xyl, 1068
xyz, 1068
xyzl, 1069
HxNonMaxSuppressionGradDir
HxNonMaxSuppressionGradDir.h, 305
HxNonMaxSuppressionGradDir.h, 305
HxNonMaxSuppressionGradDir, 305
HxNorm1
HxNorm1.h, 306
HxNorm1.h, 305
HxNorm1, 306
HxNorm2
HxNorm2.h, 306
HxNorm2.h, 306
HxNorm2, 306
HxNorm2Sqr
HxNorm2Sqr.h, 307
HxNorm2Sqr.h, 306
HxNorm2Sqr, 307
HxNormalizedCorrelation
HxNormalizedCorrelation.h, 307
HxNormalizedCorrelation.h, 307
HxNormalizedCorrelation, 307
HxNormInf
HxNormInf.h, 308
HxNormInf.h, 308
HxNormInf, 308
HxNotEqual
HxNotEqual.h, 309
HxNotEqual.h, 308
HxNotEqual, 309
HxNotEqualVal

HxNotEqualVal.h, 310
HxNotEqualVal.h, 310
HxNotEqualVal, 310
HxOFOneScale
HxOpticalFlow.h, 313
HxOpening
HxOpening.h, 311
HxOpening.h
HxOpening, 311
HxOpening.h, 311
HxOpeningByReconstruction
HxOpeningByReconstruction.h, 311
HxOpeningByReconstruction.h
HxOpeningByReconstruction, 311
HxOpeningByReconstruction.h, 311
HxOpeningByReconstructionTopHat
HxOpeningByReconstructionTopHat.h, 311
HxOpeningByReconstructionTopHat.h, 311
HxOpeningByReconstructionTopHat, 311
HxOpeningTopHat
HxOpeningTopHat.h, 312
HxOpeningTopHat.h, 312
HxOpeningTopHat, 312
HxOpponentColor
HxOpponentColor.h, 312
HxOpponentColor.h
HxOpponentColor, 312
HxOpponentColor.h, 312
HxOpticalFlow
HxOpticalFlow.h, 313
HxOpticalFlow.h
HxOpticalFlow, 313
HxOpticalFlow.h, 312
HxOFOneScale, 313
HxOpticalFlowMultiScale
HxOpticalFlowMultiScale.h, 313
HxOpticalFlowMultiScale.h, 313
HxOpticalFlowMultiScale, 313
HxOr
HxOr.h, 315
HxOr.h, 315
HxOr, 315
HxOrVal
HxOrVal.h, 317
HxOrVal.h, 316
HxOrVal, 317
HxParabolicDilation
HxParabolicDilation.h, 318
HxParabolicDilation.h, 317
HxParabolicDilation, 318
HxParabolicErosion
HxParabolicErosion.h, 319
HxParabolicErosion.h, 318
HxParabolicErosion, 319
HxPeakRemoval
HxPeakRemoval.h, 319
HxPeakRemoval.h
HxPeakRemoval, 319
HxPeakRemoval.h, 319
HxPercentile
HxPercentile.h, 320
HxPercentile.h, 319
HxPercentile, 320
HxPixelAllocator
~HxPixelAllocator, 1072
address, 1072
allocate, 1072
const_address, 1072
const_pointer, 1072
const_reference, 1072
deallocate, 1072
HxPixelAllocator, 1072
pointer, 1072
reference, 1072
size_type, 1072
value_type, 1072
HxPixelAllocator, 1072
HxPixInf
HxPixInf.h, 321
HxPixInf.h, 320
HxPixInf, 321
HxPixMax
HxPixMax.h, 321
HxPixMax.h, 321
HxPixMax, 321
HxPixMin
HxPixMin.h, 322
HxPixMin.h, 322
HxPixMin, 322
HxPixProduct
HxPixProduct.h, 323
HxPixProduct.h, 323
HxPixProduct, 323
HxPixSum
HxPixSum.h, 324
HxPixSum.h, 323
HxPixSum, 324
HxPixSup
HxPixSup.h, 324
HxPixSup.h, 324
HxPixSup, 324
HxPoint
HxPoint.h, 325
HxPoint.h, 325
HxPoint, 325
HxPointAndValue
HxPointAndValue, 1073

operator<, 1073
p, 1073
v, 1073
HxPointAndValue, 1073
HxPointInt
HxPointInt.h, 325
HxPointInt.h, 325
HxPointInt, 325
HxPointList, 1073
back_insert_iterator, 1074
eraseAll, 1074
operator<<, 1074
HxPointList.h, 326
HxPointListBackInserter, 326
HxPointListConstIter, 326
HxPointListIter, 326
HxPointListBackInserter
HxPointList.h, 326
HxPointListConstIter
HxPointList.h, 326
HxPointListIter
HxPointList.h, 326
HxPointR2
dump, 1075
HxPointR2, 1075, 1076
HxVectorR2, 1075, 1357
toString, 1075
HxPointR2, 1074
add, 1076
HxPointR2, 1075, 1076
put, 1076
sub, 1076
x, 1076
y, 1076
HxPointZ
HxPointZ, 1077, 1078
HxPointZ, 1077
HxPointZ, 1077, 1078
HxPointZList, 1078
eraseAll, 1079
operator<<, 1079
HxPolyline2d
HxPolyline2d, 1080
HxPolyline2d, 1079
~HxPolyline2d, 1081
getClosed, 1081
getNrPoints, 1081
getPoint, 1081
getPoints, 1081, 1082
HxPolyline2d, 1080
ident, 1081
put, 1082
HxPow
HxPow.h, 327
HxPow.h, 326
HxPow, 327
HxPowVal
HxPowVal.h, 328
HxPowVal.h, 328
HxPowVal, 328
HxProjectRange
HxImageRep, 641
HxProjectRange.h, 329
HxProjectRange.h, 329
HxProjectRange, 329
HxRcObject
~HxRcObject, 1083
addRef, 1083
assign, 1083
clone, 1083
doGetUnshared, 1083
getUnshared, 1083
HxRcObject, 1083
isShared, 1083
refCnt, 1083
removeRef, 1083
HxRcObject, 1082
HxRcPtr
~HxRcPtr, 1083
getUnshared, 1084
HxRcPtr, 1083
isShared, 1084
operator *, 1084
operator int, 1084
operator →, 1084
operator=, 1083
pointee, 1084
refCnt, 1084
HxRcPtr, 1083
HxRecGauss
HxRecGauss.h, 330
HxRecGauss.h, 330
HxRecGauss, 330
HxReciprocal
HxReciprocal.h, 331
HxReciprocal.h, 331
HxReciprocal, 331
HxReflect
HxReflect.h, 332
HxReflect.h, 332
HxReflect, 332
HxRegData
HxRegData, 1085, 1086
HxRegData, 1084
~HxRegData, 1086
getInt, 1087
getString, 1087
HxRegData, 1085, 1086

operator=, 1086
put, 1088
RvType, 1085
setInt, 1087
setString, 1087
toString, 1088
type, 1087
HxRegionalMaxima
HxRegionalMaxima.h, 333
HxRegionalMaxima.h, 332
HxRegionalMaxima, 333
HxRegionalMinima
HxRegionalMinima.h, 334
HxRegionalMinima.h, 334
HxRegionalMinima, 334
HxRegistry
HxRegistry, 1090
HxRegistry, 1088
~HxRegistry, 1090
eraseKey, 1092
eraseValue, 1093
exportC, 1092
exportText, 1091
findKey, 1092
findValue, 1094
getCursorKey, 1094
getCursorName, 1095
getRootKey, 1095
HxRegistry, 1090
import, 1091
insertKey, 1092
insertValue, 1093
instance, 1091
put, 1095
setCursorKey, 1094
setCursorUp, 1094
setRootKey, 1095
valueExists, 1094
HxRegistryImporter
HxRegistryImporter, 1096
HxRegistryImporter, 1095
HxRegistryImporter, 1096
HxRegKey
HxRegKeyFriend, 1098
HxRegKey, 1097
~HxRegKey, 1099
createRootKey, 1099
eraseKey, 1100
eraseValue, 1102
findKey, 1100
findValue, 1102
getInt, 1102
getKeyList, 1101
getName, 1099
getParent, 1099
getString, 1102
getValueList, 1102
insertKey, 1099
insertValue, 1101
keyListSize, 1101
put, 1103
valueListSize, 1103
HxRegKeyFriend
HxRegKey, 1098
HxRegKeyList, 1104
back_insert_iterator, 1104
HxRegKeyList.h, 335
HxRegKeyListBackInserter, 336
HxRegKeyListConstIter, 336
HxRegKeyListIter, 336
HxRegKeyPtr, 336
HxRegKeyListBackInserter
HxRegKeyList.h, 336
HxRegKeyListConstIter
HxRegKeyList.h, 336
HxRegKeyListIter
HxRegKeyList.h, 336
HxRegKeyPtr
HxRegKeyList.h, 336
HxRegValue
HxRegValue, 1105
HxRegValue, 1104
~HxRegValue, 1105
getData, 1106
getName, 1106
HxRegValue, 1105
operator<, 1106
put, 1106
setData, 1106
HxRegValueList, 1107
back_insert_iterator, 1107
HxRegValueList.h, 336
HxRegValueListBackInserter, 337
HxRegValueListConstIter, 337
HxRegValueListIter, 337
HxRegValuePtr, 337
HxRegValueListBackInserter
HxRegValueList.h, 337
HxRegValueListConstIter
HxRegValueList.h, 337
HxRegValueListIter
HxRegValueList.h, 337
HxRegValuePtr
HxRegValueList.h, 337
HxRestrict
HxImageRep, 642
HxRestrict.h, 337
HxRestrict.h, 337

HxRestrict, 337
HxRGB2Intensity
HxRGB2Intensity.h, 339
HxRGB2Intensity.h, 339
HxRGB2Intensity, 339
HxRgbBinary
HxRgbBinary, 1108
HxRgbBinary, 1107
ArithTypeDouble, 1108
className, 1109
doIt, 1108
doItDouble, 1109
HxRgbBinary, 1108
HxRgbCMY
HxRgbCMY, 1110
HxRgbCMY, 1109
ArithTypeDouble, 1110
className, 1110
doIt, 1110
doItDouble, 1110
HxRgbCMY, 1110
HxRgbDirect
HxRgbDirect, 1112
HxRgbDirect, 1111
ArithTypeDouble, 1112
className, 1112
doIt, 1112
doItDouble, 1112
HxRgbDirect, 1112
HxRgbDirectNC
HxRgbDirectNC, 1113
HxRgbDirectNC, 1112
ArithTypeDouble, 1113
className, 1114
doIt, 1114
doItDouble, 1114
HxRgbDirectNC, 1113
HxRgbHSI
HxRgbHSI, 1115
HxRgbHSI, 1114
ArithTypeDouble, 1115
className, 1116
doIt, 1115
doItDouble, 1116
HxRgbHSI, 1115
HxRgbLab
HxRgbLab, 1117
HxRgbLab, 1116
ArithTypeDouble, 1117
className, 1117
doIt, 1117
doItDouble, 1117
HxRgbLab, 1117
HxRgbLabel
HxRgbLabel, 1119
HxRgbLabel, 1118
ArithTypeDouble, 1119
className, 1119
doIt, 1119
doItDouble, 1119
HxRgbLabel, 1119
HxRgbLogMag
HxRgbLogMag, 1121
HxRgbLogMag, 1120
ArithTypeDouble, 1121
className, 1121
doIt, 1121
doItDouble, 1121
HxRgbLogMag, 1121
HxRgbLuv
HxRgbLuv, 1123
HxRgbLuv, 1122
ArithTypeDouble, 1122
className, 1123
doIt, 1123
doItDouble, 1123
HxRgbLuv, 1123
HxRgbOOO
HxRgbOOO, 1124
HxRgbOOO, 1123
ArithTypeDouble, 1124
className, 1125
doIt, 1125
doItDouble, 1125
HxRgbOOO, 1124
HxRgbStretch
HxRgbStretch, 1126
HxRgbStretch, 1125
ArithTypeDouble, 1126
className, 1127
doIt, 1127
doItDouble, 1127
HxRgbStretch, 1126
HxRgbXYZ
HxRgbXYZ, 1128
HxRgbXYZ, 1127
ArithTypeDouble, 1128
className, 1129
doIt, 1128
doItDouble, 1129
HxRgbXYZ, 1128
HxRightShift
HxRightShift.h, 340
HxRightShift.h, 339
HxRightShift, 340
HxRightShiftVal
HxRightShiftVal.h, 341
HxRightShiftVal.h, 341

HxRightShiftVal, 341
HxRotate
HxRotate.h, 342
HxRotate.h, 342
HxRotate, 342
HxRound
HxRound.h, 343
HxRound.h, 343
HxRound, 343
HxSampledBSplineCurve
HxSampledBSplineCurve, 1132
HxSampledBSplineCurve, 1129
~HxSampledBSplineCurve, 1133
AllC, 1138
allP, 1140
allSampledT, 1134
B, 1136
BAll, 1137
C, 1137
changeAllP, 1141
closestSample, 1140
continuousCurve, 1134
controlP, 1141
CPoly, 1138
curveType, 1134
dB, 1137
dBAll, 1137
dC, 1138
dCAll, 1138
dT, 1135
dTurnAngleAtC, 1139
dTurnAngleAtCAll, 1139
dump, 1141
HxSampledBSplineCurve, 1132
ident, 1133
indexOfT, 1135
intervalAffectedBy, 1136
kAtC, 1138
kAtCAll, 1139
length, 1139, 1140
makeInterpolating, 1133
makeUniform, 1133
nSamples, 1134
numP, 1140
PThatAffectSample, 1136
sampledInterval, 1135
sampledT, 1134
samplesAffectedBy, 1136
samplingAlg, 1134
translateCurve, 1141
HxSampledBSplineInterval
HxSampledBSplineInterval, 1143
HxSampledBSplineInterval, 1142
~HxSampledBSplineInterval, 1143
begin, 1144
contains, 1144
end, 1144
HxSampledBSplineInterval, 1143
middle, 1144
next, 1144
ratio, 1144
size, 1145
HxScalarDouble
HxScalarDouble, 1151
operator new, 1150
operator=, 1150
setValue, 1146
HxScalarDouble, 1145
abs, 1154
acos, 1156
and, 1161
asin, 1156
atan, 1157
atan2, 1157
ceil, 1154
complement, 1154
cos, 1156
cosh, 1157
cross, 1162
dim, 1151
dot, 1162
exp, 1157
floor, 1154
getValue, 1151
HxScalarDouble, 1151
inf, 1160
infAssign, 1160
LARGE_VAL, 1163
leftShift, 1161
log, 1158
log10, 1158
max, 1155, 1159
maxAssign, 1159
min, 1155, 1159
minAssign, 1159
mod, 1161
norm1, 1155
norm2, 1155
normInf, 1155
operator $*$, 1163
operator $*=$, 1158
operator HxComplex, 1152
operator HxScalarInt, 1151
operator HxVec2Double, 1152
operator HxVec2Int, 1152
operator HxVec3Double, 1152
operator HxVec3Int, 1152
operator!=, 1153

operator+, 1162
operator+=, 1158
operator-, 1153, 1163
operator-=, 1158
operator/, 1163
operator/=, 1159
operator<, 1153
operator<=, 1153
operator==, 1152
operator>, 1153
operator>=, 1153
or, 1161
pow, 1160
product, 1155
put, 1162
rightShift, 1161
round, 1154
sin, 1156
sinh, 1157
SMALL_VAL, 1163
sqrt, 1156
sum, 1154
sup, 1160
supAssign, 1160
tan, 1156
tanh, 1157
toString, 1162
x, 1151
xor, 1161
HxScalarDoubleValue
HxValue, 1257
HxScalarInt
HxScalarInt, 1169
operator new, 1169
setValue, 1164
HxScalarInt, 1164
abs, 1172
acos, 1175
and, 1179
asin, 1175
atan, 1175
atan2, 1175
ceil, 1172
complement, 1172
cos, 1174
cosh, 1176
cross, 1180
dim, 1169
dot, 1180
exp, 1176
floor, 1173
getValue, 1170
HxScalarInt, 1169
inf, 1178
infAssign, 1178
LARGE_VAL, 1181
leftShift, 1180
log, 1176
log10, 1176
max, 1173, 1178
maxAssign, 1178
min, 1173, 1177
minAssign, 1177
mod, 1179
norm1, 1174
norm2, 1174
normInf, 1174
operator *, 1181
operator *=, 1177
operator HxComplex, 1171
operator HxScalarDouble, 1170
operator HxVec2Double, 1170
operator HxVec2Int, 1170
operator HxVec3Double, 1171
operator HxVec3Int, 1170
operator!=, 1171
operator+, 1181
operator+=, 1176
operator-, 1172, 1181
operator-=, 1177
operator/, 1181
operator/=, 1177
operator<, 1171
operator<=, 1171
operator==, 1171
operator>, 1172
operator>=, 1172
or, 1179
pow, 1179
product, 1173
put, 1180
rightShift, 1180
round, 1173
sin, 1174
sinh, 1175
SMALL_VAL, 1181
sqrt, 1174
sum, 1173
sup, 1178
supAssign, 1179
tan, 1175
tanh, 1176
toString, 1180
x, 1170
xor, 1179
HxScalarIntValue
HxValue, 1257
HxScale

HxScale.h, 344
HxScale.h, 343
HxScale, 344
HxSegmentation2d
HxSegmentation2d, 1183
HxSegmentation2d, 1182
~HxSegmentation2d, 1183
addBlob, 1184, 1185
addRelation, 1185
getBlobBegin, 1185
getBlobEnd, 1185
getBlobInserter, 1185
getInputImage, 1184
getLabeledImage, 1184
getRelation, 1185
HxSegmentation2d, 1183
ident, 1184
put, 1186
setInputImage, 1184
setLabeledImage, 1184
HxSegmentationCentralMoments
HxSegmentationCentralMoments.h, 344
HxSegmentationCentralMoments.h, 344
HxSegmentationCentralMoments, 344
HxSegmentationHistogram
HxSegmentationHistogram.h, 345
HxSegmentationHistogram.h, 345
HxSegmentationHistogram, 345
HxSegmentationMean
HxSegmentationMean.h, 346
HxSegmentationMean.h, 345
HxSegmentationMean, 346
HxSegmentationMedian
HxSegmentationMedian.h, 346
HxSegmentationMedian.h, 346
HxSegmentationMedian, 346
HxSegmentationMoments
HxSegmentationMoments.h, 347
HxSegmentationMoments.h, 346
HxSegmentationMoments, 347
HxSegmentationStDev
HxSegmentationStDev.h, 347
HxSegmentationStDev.h, 347
HxSegmentationStDev, 347
HxSegmentationSum
HxSegmentationSum.h, 348
HxSegmentationSum.h, 348
HxSegmentationSum, 348
HxSegmentationVariance
HxSegmentationVariance.h, 348
HxSegmentationVariance.h, 348
HxSegmentationVariance, 348
HxSetBorderValue
HxSetBorderValue.h, 349
HxSetBorderValue.h, 349
HxSetBorderValue, 349
HxSetPartImage
HxSetPartImage.c, 350
HxSetPartImage.h, 350
HxSetPartImage.c, 349
HxSetPartImage, 350
HxSetPartImage.h, 350
HxSetPartImage, 350
HxSF
~HxSF, 1187
erodeSF, 1186
getConnectivity, 1186
getHorizontalKernel, 1186
getKernel, 1186
getVerticalKernel, 1186
HxSF, 1187, 1188
HxSFFactory, 1187
isSeparable, 1186
isSymetric, 1186
HxSF, 1186
dilateSF, 1188
HxSF, 1188
operator=, 1188
rotateSF, 1189
HxSFFactory
HxSF, 1187
makeBoxSF, 1189
makeCrossSF, 1189
makeDiamondSF, 1189
makeDiskSF, 1189
makeGaussianSF, 1189
makeParabolaSF, 1189
HxSFFactory, 1189
fromFunction, 1190
instance, 1190
makeFlatSF, 1190
makeSFfromImage, 1190
HxSin
HxSin.h, 351
HxSin.h, 351
HxSin, 351
HxSinh
HxSinh.h, 352
HxSinh.h, 351
HxSinh, 352
HxSizes
HxSizes.h, 352
HxSizes.h
ClassName, 352
makeString, 352
HxSizes.h, 352
HxSizes, 352
HxSkeleton

HxSkeleton.h, 353
HxSkeleton.h, 353
HxSkeleton, 353
HxSKIZ
HxSKIZ.h, 354
HxSKIZ.h, 353
HxSKIZ, 354
HxSqrt
HxSqrt.h, 355
HxSqrt.h, 354
HxSqrt, 355
HxSquaredDistance
HxSquaredDistance.h, 356
HxSquaredDistance.h, 355
HxSquaredDistance, 356
HxString
HxStringNative.h, 358
HxStringList
HxStringList, 1191
HxStringList, 1191
back_insert_iterator, 1192
eraseAll, 1192
operator<<, 1192
HxStringList.h
makeString, 357
splitString, 357
HxStringList.h, 357
HxStringListBackInserter, 357
HxStringListConstIter, 357
HxStringListIter, 357
HxStringListBackInserter
HxStringList.h, 357
HxStringListConstIter
HxStringList.h, 357
HxStringListIter
HxStringList.h, 357
HxStringNative.h
atof, 358
atoi, 358
atol, 358
ClassName, 358
makeString, 358
HxStringNative.h, 358
HxString, 358
HxSub
HxSub.h, 359
HxSub.h, 358
HxSub, 359
HxSubSat
HxSubSat.h, 360
HxSubSat.h, 360
HxSubSat, 360
HxSubVal
HxSubVal.h, 361
HxSubVal.h, 361
HxSubVal, 361
HxSup
HxSup.h, 362
HxSup.h, 362
HxSup, 362
HxSupremumReconstruction
HxSupremumReconstruction.h, 363
HxSupremumReconstruction.h, 363
HxSupremumReconstruction, 363
HxSupVal
HxSupVal.h, 364
HxSupVal.h, 364
HxSupVal, 364
HxTag
HxTag, 1193
HxTag, 1192
~HxTag, 1193
clone, 1194
getName, 1194
HxTag, 1193
put, 1194
HxTag1Phase, 1194
toString, 1194
HxTag2Phase, 1195
toString, 1195
HxTagCnum, 1195
toString, 1196
HxTagIsSet
HxTagList.h, 366
HxTagList
HxTagList, 1197
HxTagList, 1196
~HxTagList, 1197
addTag, 1198
erase, 1198
getTag, 1198
HxTagList, 1197
List, 1197
operator=, 1198
put, 1199
TagPtr, 1197
toString, 1199
HxTagList.h
operator<<, 365
HxTagList.h, 365
HxAddTag, 366
HxGetTag, 366
HxMakeTagList, 366
HxTagIsSet, 366
HxTagLoop, 1199
toString, 1200
HxTagNPhase, 1200
toString, 1200

HxTagPixOpIn, 1201
toString, 1201
HxTagPixOpOut, 1201
toString, 1202
HxTagTem
HxTagTem, 1203
HxTagTem, 1202
∼HxTagTem, 1203
clone, 1203
getValue, 1203
HxTagTem, 1203
put, 1203
HxTagTransInVar, 1204
toString, 1204
HxTagTransVar, 1204
toString, 1205
HxTan
HxTan.h, 367
HxTan.h, 367
HxTan, 367
HxTanh
HxTanh.h, 368
HxTanh.h, 367
HxTanh, 368
HxThickening
HxThickening.h, 368
HxThickening.h, 368
HxThickening, 368
HxThinning
HxThinning.h, 369
HxThinning.h, 369
HxThinning, 369
HxThreshold
HxThreshold.h, 370
HxThreshold.h, 370
HxThreshold, 370
HxTranslate
HxTranslate.h, 371
HxTranslate.h, 371
HxTranslate, 371
HxTranspose
HxTranspose.c, 372
HxTranspose.h, 373
HxTranspose.c, 371
HxTranspose, 372
HxTranspose_Line, 372
HxTranspose.h, 373
HxTranspose, 373
HxTranspose_Line
HxTranspose.c, 372
HxTriStateThreshold
HxTriStateThreshold.h, 374
HxTriStateThreshold.h, 373
HxTriStateThreshold, 374
HxUnaryMax
HxUnaryMax.h, 375
HxUnaryMax.h, 374
HxUnaryMax, 375
HxUnaryMin
HxUnaryMin.h, 375
HxUnaryMin.h, 375
HxUnaryMin, 375
HxUnaryProduct
HxUnaryProduct.h, 376
HxUnaryProduct.h, 376
HxUnaryProduct, 376
HxUnarySum
HxUnarySum.h, 376
HxUnarySum.h, 376
HxUnarySum, 376
HxUniform
HxUniform.h, 377
HxUniform.h, 377
HxUniform, 377
HxUniformNonSep
HxUniformNonSep.h, 378
HxUniformNonSep.h, 378
HxUniformNonSep, 378
HxUpoAbs
HxUpoAbs, 1206
HxUpoAbs, 1205
className, 1206
doIt, 1206
HxUpoAbs, 1206
TransVarianceCategory, 1206
HxUpoAcos
HxUpoAcos, 1207
HxUpoAcos, 1206
className, 1208
doIt, 1207
HxUpoAcos, 1207
TransVarianceCategory, 1207
HxUpoArg
HxUpoArg, 1209
HxUpoArg, 1208
className, 1209
doIt, 1209
HxUpoArg, 1209
TransVarianceCategory, 1209
HxUpoAsin
HxUpoAsin, 1210
HxUpoAsin, 1209
className, 1210
doIt, 1210
HxUpoAsin, 1210
TransVarianceCategory, 1210
HxUpoAtan
HxUpoAtan, 1212

HxUpoAtan, 1211
classNameHxUpoAtan, 1212
doIt, 1212
HxUpoAtan, 1212
TransVarianceCategory, 1212
HxUpoAtan2
HxUpoAtan2, 1213
HxUpoAtan2, 1212
classNameHxUpoAtan2, 1213
doIt, 1213
HxUpoAtan2, 1213
TransVarianceCategory, 1213
HxUpoCeil
HxUpoCeil, 1215
HxUpoCeil, 1214
classNameHxUpoCeil, 1215
doIt, 1215
HxUpoCeil, 1215
TransVarianceCategory, 1215
HxUpoColSpace
HxUpoColSpace, 1216
HxUpoColSpace, 1215
classNameHxUpoColSpace, 1216
doIt, 1216
HxUpoColSpace, 1216
TransVarianceCategory, 1216
HxUpoComplement
HxUpoComplement, 1218
HxUpoComplement, 1217
classNameHxUpoComplement, 1218
doIt, 1218
HxUpoComplement, 1218
TransVarianceCategory, 1218
HxUpoConjugate
HxUpoConjugate, 1219
HxUpoConjugate, 1218
classNameHxUpoConjugate, 1219
doIt, 1219
HxUpoConjugate, 1219
TransVarianceCategory, 1219
HxUpoCos
HxUpoCos, 1221
HxUpoCos, 1220
classNameHxUpoCos, 1221
doIt, 1221
HxUpoCos, 1221
TransVarianceCategory, 1221
HxUpoCosh
HxUpoCosh, 1222
HxUpoCosh, 1221
classNameHxUpoCosh, 1222
doIt, 1222
HxUpoCosh, 1222
TransVarianceCategory, 1222
HxUpoExp
HxUpoExp, 1224
HxUpoExp, 1223
classNameHxUpoExp, 1224
doIt, 1224
HxUpoExp, 1224
TransVarianceCategory, 1224
HxUpoFloor
HxUpoFloor, 1225
HxUpoFloor, 1224
classNameHxUpoFloor, 1225
doIt, 1225
HxUpoFloor, 1225
TransVarianceCategory, 1225
HxUpoLog
HxUpoLog, 1227
HxUpoLog, 1226
classNameHxUpoLog, 1227
doIt, 1227
HxUpoLog, 1227
TransVarianceCategory, 1227
HxUpoLog10
HxUpoLog10, 1228
HxUpoLog10, 1227
classNameHxUpoLog10, 1228
doIt, 1228
HxUpoLog10, 1228
TransVarianceCategory, 1228
HxUpoMax
HxUpoMax, 1230
HxUpoMax, 1229
classNameHxUpoMax, 1230
doIt, 1230
HxUpoMax, 1230
TransVarianceCategory, 1230
HxUpoMin
HxUpoMin, 1231
HxUpoMin, 1230
classNameHxUpoMin, 1231
doIt, 1231
HxUpoMin, 1231
TransVarianceCategory, 1231
HxUpoNegate
HxUpoNegate, 1233
HxUpoNegate, 1232
classNameHxUpoNegate, 1233
doIt, 1233
HxUpoNegate, 1233
TransVarianceCategory, 1233
HxUpoNorm1
HxUpoNorm1, 1234
HxUpoNorm1, 1233
classNameHxUpoNorm1, 1234
doIt, 1234

HxUpoNorm1, 1234
TransVarianceCategory, 1234
HxUpoNorm2
HxUpoNorm2, 1236
HxUpoNorm2, 1235
className, 1236
doIt, 1236
HxUpoNorm2, 1236
TransVarianceCategory, 1236
HxUpoNorm2Sqr
HxUpoNorm2Sqr, 1237
HxUpoNorm2Sqr, 1236
className, 1237
doIt, 1237
HxUpoNorm2Sqr, 1237
TransVarianceCategory, 1237
HxUpoNormInf
HxUpoNormInf, 1239
HxUpoNormInf, 1238
className, 1239
doIt, 1239
HxUpoNormInf, 1239
TransVarianceCategory, 1239
HxUpoProduct
HxUpoProduct, 1240
HxUpoProduct, 1239
className, 1240
doIt, 1240
HxUpoProduct, 1240
TransVarianceCategory, 1240
HxUpoRound
HxUpoRound, 1242
HxUpoRound, 1241
className, 1242
doIt, 1242
HxUpoRound, 1242
TransVarianceCategory, 1242
HxUpoSetPartImageInst
HxUpoSetPartImageInst.c, 379
HxUpoSetPartImageInst.c, 378
HxUpoSetPartImageInst, 379
HxUpoSin
HxUpoSin, 1243
HxUpoSin, 1242
className, 1243
doIt, 1243
HxUpoSin, 1243
TransVarianceCategory, 1243
HxUpoSinh
HxUpoSinh, 1245
HxUpoSinh, 1244
className, 1245
doIt, 1245
HxUpoSinh, 1245
TransVarianceCategory, 1245
HxUpoSqrt
HxUpoSqrt, 1246
HxUpoSqrt, 1245
className, 1246
doIt, 1246
HxUpoSqrt, 1246
TransVarianceCategory, 1246
HxUpoSum
HxUpoSum, 1248
HxUpoSum, 1247
className, 1248
doIt, 1248
HxUpoSum, 1248
TransVarianceCategory, 1248
HxUpoTan
HxUpoTan, 1249
HxUpoTan, 1248
className, 1249
doIt, 1249
HxUpoTan, 1249
TransVarianceCategory, 1249
HxUpoTanh
HxUpoTanh, 1251
HxUpoTanh, 1250
className, 1251
doIt, 1251
HxUpoTanh, 1251
TransVarianceCategory, 1251
HxUpoTriStateThreshold
HxUpoTriStateThreshold, 1252
HxUpoTriStateThreshold, 1251
className, 1253
doIt, 1252
HxUpoTriStateThreshold, 1252
TransVarianceCategory, 1252
HxValleyRemoval
HxValleyRemoval.h, 379
HxValleyRemoval.h
HxValleyRemoval, 379
HxValleyRemoval.h, 379
HxValue
HxValue, 1255–1257
operator<<, 1255
toString, 1255
HxValue, 1253
HxComplexValue, 1258
HxScalarDoubleValue, 1257
HxScalarIntValue, 1257
HxValue, 1255–1257
HxVec2DoubleValue, 1258
HxVec2IntValue, 1258
HxVec3DoubleValue, 1258
HxVec3IntValue, 1258

operator HxComplex, 1261
operator HxScalarDouble, 1259
operator HxScalarInt, 1259
operator HxVec2Double, 1260
operator HxVec2Int, 1259
operator HxVec3Double, 1260
operator HxVec3Int, 1260
Tag, 1255
tag, 1257
HxValueList, 1261
back_insert_iterator, 1262
eraseAll, 1262
operator<<, 1262
HxValueList.h, 379
HxValueListBackInserter, 380
HxValueListConstIter, 380
HxValueListIter, 380
HxValueListBackInserter
HxValueList.h, 380
HxValueListConstIter
HxValueList.h, 380
HxValueListIter
HxValueList.h, 380
HxValueType
HxValueType.h, 381
HxValueType.h
HxValueType_put, 381
makeString, 381
operator<<, 381
HxValueType.h, 380
HxValueType, 381
HxValueType_put
HxValueType.h, 381
HxVec2Byte
HxVec2Byte.h, 381
HxVec2Byte.h
dummyVec2Byte, 381
HxVec2Byte.h, 381
HxVec2Byte, 381
HxVec2Double
HxVec2Double, 1268
operator new, 1268
operator=, 1268
setValue, 1263
HxVec2Double, 1262
abs, 1271
acos, 1274
and, 1278
asin, 1274
atan, 1274
atan2, 1274
ceil, 1271
complement, 1271
cos, 1273
cosh, 1275
cross, 1279
dim, 1268
dot, 1279
exp, 1275
floor, 1272
getValue, 1269
HxVec2Double, 1268
inf, 1277
infAssign, 1277
LARGE_VAL, 1281
leftShift, 1279
log, 1275
log10, 1275
max, 1272, 1277
maxAssign, 1277
min, 1272, 1276
minAssign, 1276
mod, 1278
norm1, 1273
norm2, 1273
normInf, 1273
operator *, 1280
operator *=, 1276
operator HxComplex, 1270
operator HxScalarDouble, 1269
operator HxScalarInt, 1269
operator HxVec2Int, 1269
operator HxVec3Double, 1270
operator HxVec3Int, 1269
operator!=, 1270
operator+, 1280
operator+=, 1275
operator-, 1271, 1280
operator-=, 1276
operator/, 1280
operator/=, 1276
operator<, 1270
operator<=, 1270
operator==, 1270
operator>, 1271
operator>=, 1271
or, 1278
pow, 1278
product, 1272
put, 1279
rightShift, 1279
round, 1272
sin, 1273
sinh, 1274
SMALL_VAL, 1281
sqrt, 1273
sum, 1272
sup, 1277

supAssign, 1278
tan, 1274
tanh, 1275
toString, 1279
x, 1268
xor, 1278
y, 1269
HxVec2DoubleValue
HxValue, 1258
HxVec2Float
HxVec2Float.h, 382
HxVec2Float.h
dummyVec2Float, 382
HxVec2Float.h, 381
HxVec2Float, 382
HxVec2Int
HxVec2Int, 1286, 1287
operator new, 1286
setValue, 1282
HxVec2Int, 1281
abs, 1290
acos, 1292
and, 1297
asin, 1292
atan, 1293
atan2, 1293
ceil, 1290
complement, 1290
cos, 1292
cosh, 1293
cross, 1298
dim, 1287
dot, 1298
exp, 1293
floor, 1290
getValue, 1287
HxVec2Int, 1286, 1287
inf, 1296
infAssign, 1296
LARGE_VAL, 1299
leftShift, 1297
log, 1294
log10, 1294
max, 1291, 1295
maxAssign, 1295
min, 1291, 1295
minAssign, 1295
mod, 1297
norm1, 1291
norm2, 1291
normInf, 1291
operator ∗, 1299
operator ∗=, 1294
operator HxComplex, 1288
operator HxScalarDouble, 1288
operator HxScalarInt, 1287
operator HxVec2Double, 1288
operator HxVec3Double, 1288
operator HxVec3Int, 1288
operator!=, 1289
operator+, 1299
operator+=, 1294
operator-, 1289, 1299
operator-=, 1294
operator/, 1299
operator/=, 1295
operator<, 1289
operator<=, 1289
operator==, 1288
operator>, 1289
operator>=, 1289
or, 1297
pow, 1296
product, 1291
put, 1298
rightShift, 1298
round, 1290
sin, 1292
sinh, 1293
SMALL_VAL, 1299
sqrt, 1292
sum, 1290
sup, 1296
supAssign, 1296
tan, 1292
tanh, 1293
toString, 1298
x, 1287
xor, 1297
y, 1287
HxVec2IntValue
HxValue, 1258
HxVec2Short
HxVec2Short.h, 382
HxVec2Short.h
dummyVec2Short, 382
HxVec2Short.h, 382
HxVec2Short, 382
HxVec2Tem
HxVec2Tem, 1300
operator ∗, 1300
operator ∗=, 1300
operator double, 1300
operator HxVec2Double, 1300
operator HxVec2Int, 1300
operator int, 1300
operator new, 1300
operator+, 1300

operator+=, 1300
operator-, 1300
operator-=, 1300
operator/, 1300
operator/=, 1300
put, 1300
value, 1300
x, 1300
y, 1300
HxVec2Tem, 1300
HxVec3Byte
HxVec3Byte.h, 383
HxVec3Byte.h
dummyVec3Byte, 383
HxVec3Byte.h, 382
HxVec3Byte, 383
HxVec3Double
HxVec3Double, 1306, 1307
operator new, 1306
setValue, 1301
HxVec3Double, 1301
abs, 1310
acos, 1313
and, 1318
asin, 1313
atan, 1313
atan2, 1313
ceil, 1310
complement, 1310
cos, 1312
cosh, 1314
cross, 1319
dim, 1307
dot, 1318
exp, 1314
floor, 1310
getValue, 1307
HxVec3Double, 1306, 1307
inf, 1316
infAssign, 1317
LARGE_VAL, 1320
leftShift, 1318
log, 1314
log10, 1315
max, 1311, 1316
maxAssign, 1316
min, 1311, 1316
minAssign, 1316
mod, 1317
norm1, 1311
norm2, 1312
normInf, 1312
operator ∗, 1320
operator ∗=, 1315
operator HxComplex, 1308
operator HxScalarDouble, 1308
operator HxScalarInt, 1308
operator HxVec2Double, 1308
operator HxVec2Int, 1308
operator HxVec3Int, 1308
operator!=, 1309
operator+, 1319
operator+=, 1315
operator-, 1310, 1319
operator-=, 1315
operator/, 1320
operator/=, 1315
operator<, 1309
operator<=, 1309
operator==, 1309
operator>, 1309
operator>=, 1309
or, 1318
pow, 1317
product, 1311
put, 1319
rightShift, 1318
round, 1310
sin, 1312
sinh, 1314
SMALL_VAL, 1320
sqrt, 1312
sum, 1311
sup, 1317
supAssign, 1317
tan, 1313
tanh, 1314
toString, 1319
x, 1307
xor, 1318
y, 1307
z, 1307
HxVec3DoubleValue
HxValue, 1258
HxVec3Float
HxVec3Float.h, 383
HxVec3Float.h
dummyVec3Float, 383
HxVec3Float.h, 383
HxVec3Float, 383
HxVec3Int
HxVec3Int, 1326, 1327
operator new, 1326
setValue, 1321
HxVec3Int, 1321
abs, 1330
acos, 1333
and, 1337

asin, 1333
atan, 1333
atan2, 1333
ceil, 1330
complement, 1330
cos, 1332
cosh, 1334
cross, 1339
dim, 1327
dot, 1338
exp, 1334
floor, 1330
getValue, 1327
HxVec3Int, 1326, 1327
inf, 1336
infAssign, 1336
LARGE_VAL, 1340
leftShift, 1338
log, 1334
log10, 1334
max, 1331, 1336
maxAssign, 1336
min, 1331, 1335
minAssign, 1336
mod, 1337
norm1, 1331
norm2, 1331
normInf, 1332
operator ∗, 1340
operator ∗=, 1335
operator HxComplex, 1328
operator HxScalarDouble, 1328
operator HxScalarInt, 1328
operator HxVec2Double, 1328
operator HxVec2Int, 1328
operator HxVec3Double, 1328
operator!=, 1329
operator+, 1339
operator+=, 1335
operator-, 1330, 1339
operator-=, 1335
operator/, 1340
operator/=, 1335
operator<, 1329
operator<=, 1329
operator==, 1329
operator>, 1329
operator>=, 1329
or, 1338
pow, 1337
product, 1331
put, 1339
rightShift, 1338
round, 1330
sin, 1332
sinh, 1333
SMALL_VAL, 1340
sqrt, 1332
sum, 1331
sup, 1337
supAssign, 1337
tan, 1332
tanh, 1334
toString, 1339
x, 1327
xor, 1338
y, 1327
z, 1327
HxVec3IntValue
HxValue, 1258
HxVec3Short
HxVec3Short.h, 384
HxVec3Short.h
dummyVec3Short, 384
HxVec3Short.h, 383
HxVec3Short, 384
HxVec3Tem
HxVec3Tem, 1341
operator ∗, 1341
operator ∗=, 1341
operator double, 1341
operator HxVec3Double, 1341
operator HxVec3Int, 1341
operator int, 1341
operator new, 1341
operator+, 1341
operator+=, 1341
operator-, 1341
operator-=, 1341
operator/, 1341
operator/=, 1341
put, 1341
value, 1341
x, 1341
y, 1341
z, 1341
HxVec3Tem, 1341
HxVector
∼HxVector, 1345
HxMatrix, 984, 1345
HxVector, 1345–1347
put, 1345
HxVector, 1342
abs, 1352
add, 1349
cos, 1351
cosh, 1351
diag, 1349

div, 1350
exp, 1351
HxVector, 1345–1347
log, 1352
map, 1352
mul, 1350
nElem, 1347
operator ∗, 1353
operator!=, 1356
operator+, 1354
operator-, 1348, 1355
operator/, 1353, 1354
operator=, 1347, 1348
operator==, 1356
operator[], 1348
sgn, 1352
sin, 1350
sinh, 1351
sqrt, 1352
sub, 1349
t, 1348
tan, 1351
tanh, 1351
valid, 1347
HxVectorR2
dump, 1357
HxPointR2, 1075, 1357
HxVectorR2, 1358
toString, 1357
HxVectorR2, 1356
add, 1358
cross2D, 1359
div, 1359
dot, 1359
HxVectorR2, 1358
magnitude, 1359
mul, 1359
normal, 1360
put, 1360
squaredMagnitude, 1359
sub, 1358
x, 1358
y, 1358
HxVectorToImageRep
HxMatrixConv.h, 293
HxWatershed
HxWatershed.h, 384
HxWatershed.h, 384
HxWatershed, 384
HxWatershedMarkers
HxWatershedMarkers.h, 385
HxWatershedMarkers.h, 385
HxWatershedMarkers, 385
HxWatershedMarkers2
HxWatershedMarkers2.h, 386
HxWatershedMarkers2.h, 386
HxWatershedMarkers2, 386
HxWatershedSlow
HxWatershedSlow.h, 387
HxWatershedSlow.h, 387
HxWatershedSlow, 387
HxWeightMaskSum
HxWeightMaskSum.h, 394
HxWeightMaskSum.h, 394
HxWeightMaskSum, 394
HxWriteFile
HxWriteFile.h, 395
HxWriteFile.h, 394
HxWriteFile, 395
HxXor
HxXor.h, 396
HxXor.h, 396
HxXor, 396
HxXorVal
HxXorVal.h, 397
HxXorVal.h, 397
HxXorVal, 397

i
HxMatrix, 995
ident
HxBlob2d, 404
HxBlob2dRelation, 412
HxBSplineCurve, 481
HxHistogram, 556
HxImageData, 588
HxImageRep, 627
HxImageSeq, 650
HxImageSeqData, 655
HxImageSignature, 688
HxNJet, 1067
HxPolyline2d, 1081
HxSampledBSplineCurve, 1133
HxSegmentation2d, 1184
imageDimensionality
HxImageSignature, 691
imagesFromFile
HxImageFactory, 615
imagesToFile
HxImageFactory, 614
Img1DataPtrType
HxImgFtorI2Cast, 748
HxImgFtorI3Cast, 763
HxImgFtorI4Cast, 774
Img2DataPtrType
HxImgFtorI2Cast, 749
HxImgFtorI3Cast, 763
HxImgFtorI4Cast, 774

Img3DataPtrType
HxImgFtorI3Cast, 763
HxImgFtorI4Cast, 774
Img4DataPtrType
HxImgFtorI4Cast, 774
ImgDataPtrType
HxImgFtorI1Cast, 733
import
HxImageData, 589, 590
HxRegistry, 1091
inc
HxCnum, 498
incBin
HxHistogram, 564
incBinChecked
HxHistogram, 560, 561
includes
HxBoundingBox, 417
incX
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 527, 528
HxDataPtr3dScalarTem, 528
incXYZ
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529
incY
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 527, 528
HxDataPtr3dScalarTem, 528
incZ
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 527
HxDataPtr3dScalarTem, 528
indexOfT
HxSampledBSplineCurve, 1135
inf
HxComplex, 522
HxScalarDouble, 1160
HxScalarInt, 1178
HxVec2Double, 1277
HxVec2Int, 1296
HxVec3Double, 1316
HxVec3Int, 1336
infAssign
HxComplex, 522
HxScalarDouble, 1160
HxScalarInt, 1178
HxVec2Double, 1277
HxVec2Int, 1296
HxVec3Double, 1317
HxVec3Int, 1336
init
HxExportExtraIdentMaskCentralMoments, 533
HxKerNgbNormCorrelation, 975
HxNgbBernsen, 1039
HxNgbDefuz, 1040
HxNgbHilditch, 1042
HxNgbIsMaxGradDir2d, 1046
HxNgbKuwahara, 1048
HxNgbLocalMode, 1051
HxNgbLWshed2d, 1054
HxNgbNonMaxSuppression2d, 1059
HxNgbPercentile2d, 1062
init2
HxKerNgbNormCorrelation, 975
inout
HxImageData, 591
insert
HxImgFtorTable, 857
HxNameTable, 1037
inserted
HxImgFtorObserver, 822
insertKey
HxRegistry, 1092
HxRegKey, 1099
insertKnot
HxBSplineBasis, 476
HxBSplineCurve, 489
insertVal
HxHistogram, 561–563
insertValChecked
HxHistogram, 559, 560
insertValue
HxRegistry, 1093
HxRegKey, 1101
instance
HxImageFactory, 607
HxImgFtorKeyNameTable, 803
HxImgFtorRuleBase, 843
HxImgFtorTable, 857
HxRegistry, 1091
HxSFFactory, 1190
intersect
HxBoundingBox, 416
intersection
HxHistogram, 569
intervalAffectedBy
HxSampledBSplineCurve, 1136
inverseProjectDomain
HxImageData, 586
HxImageTem2d, 697
HxImageTem3d, 699
inverseProjectRange
HxImageData, 586
isClosed

HxBSplineInterval, 494
isColor
HxNJet, 1068
isEmpty
HxBoundingBox, 416
isEqual
HxImageSignature, 689
isNull
HxHistogram, 555
HxImageRep, 627
HxImageSeq, 650
VideoReader, 1363
isSeparable
HxSF, 1186
isShared
HxRcObject, 1083
HxRcPtr, 1084
isSymetric
HxSF, 1186
iterator
HxImageList, 617
IteratorCategory
HxKerNgbNormCorrelation, 974
HxNgbBernsen, 1039
HxNgbDefuz, 1041
HxNgbHilditch, 1043
HxNgbIsMaxGradDir2d, 1045
HxNgbKuwahara, 1048
HxNgbLocalMode, 1050
HxNgbLWshed2d, 1054
HxNgbNonMaxSuppression2d, 1057
HxNgbPercentile2d, 1061

kAtC
HxBSplineCurve, 485
HxSampledBSplineCurve, 1138
kAtCAll
HxSampledBSplineCurve, 1139
kernel
HxMfGenConv, 1015
HxMfKernelNgb, 1020
HxMfQueueBased, 1031
kernel2
HxMfGenConv, 1015
kernel3
HxMfGenConv, 1016
keyListSize
HxRegKey, 1101
KeyType
HxImgFtorBpo, 701
HxImgFtorDiy, 706
HxImgFtorExportExtra, 709
HxImgFtorGenConv2d, 713
HxImgFtorGenConv2dK1d, 715
HxImgFtorGenConv2dSep, 718
HxImgFtorGenConv3d, 722
HxImgFtorGenConv3dK1d, 725
HxImgFtorI1, 729
HxImgFtorI1Cast, 732
HxImgFtorI2, 738
HxImgFtorI2Cast, 745
HxImgFtorI3, 756
HxImgFtorI3Cast, 761
HxImgFtorI4, 770
HxImgFtorI4Cast, 773
HxImgFtorIM, 779
HxImgFtorIMCast, 781
HxImgFtorIMN, 785
HxImgFtorIMNCast, 788
HxImgFtorInOut, 793
HxImgFtorKernelNgb2d, 796
HxImgFtorMNpo, 805
HxImgFtorMpo, 808
HxImgFtorNgb2d, 812
HxImgFtorNgb2dExtra, 814
HxImgFtorNgb2dExtra2, 817
HxImgFtorRecGenConv2d, 829
HxImgFtorRecGenConv2dK1d, 831
HxImgFtorRgb2d, 836
HxImgFtorRgb3d, 838
HxImgFtorSet, 848
HxImgFtorSetBorder2d, 850
HxImgFtorSetBorder3d, 853
HxImgFtorUpo, 860
killThisOne
QThinning, 1361
knot
HxBSplineBasis, 474
knotsType
HxBSplineBasis, 473

LARGE_VAL
HxComplex, 526
HxScalarDouble, 1163
HxScalarInt, 1181
HxVec2Double, 1281
HxVec2Int, 1299
HxVec3Double, 1320
HxVec3Int, 1340
leftShift
HxComplex, 524
HxScalarDouble, 1161
HxScalarInt, 1180
HxVec2Double, 1279
HxVec2Int, 1297
HxVec3Double, 1318
HxVec3Int, 1338
length

HxBSplineCurve, 485, 486
HxBSplineInterval, 493
HxSampledBSplineCurve, 1139, 1140
lift2dTo3dXY
HxMatrix, 993
List
HxTagList, 1197
localPixelInit
QThinning, 1361
log
HxComplex, 520
HxMatrix, 1000
HxScalarDouble, 1158
HxScalarInt, 1176
HxVec2Double, 1275
HxVec2Int, 1294
HxVec3Double, 1314
HxVec3Int, 1334
HxVector, 1352
log10
HxComplex, 520
HxScalarDouble, 1158
HxScalarInt, 1176
HxVec2Double, 1275
HxVec2Int, 1294
HxVec3Double, 1315
HxVec3Int, 1334
lowBin
HxHistogram, 557

magnitude
HxVectorR2, 1359
makeBoxSF
HxSFFactory, 1189
makeCrossSF
HxSFFactory, 1189
makeDiamondSF
HxSFFactory, 1189
makeDiskSF
HxSFFactory, 1189
makeFlatSF
HxSFFactory, 1190
makeGaussianSF
HxSFFactory, 1189
makeInterpolating
HxBSplineCurve, 481
HxSampledBSplineCurve, 1133
makeParabolaSF
HxSFFactory, 1189
makeScratch
HxImageTem, 695
makeSFfromImage
HxSFFactory, 1190
makeString
HxGeoIntType.h, 231
HxSizes.h, 352
HxStringList.h, 357
HxStringNative.h, 358
HxValueType.h, 381
makeUniform
HxBSplineCurve, 480
HxSampledBSplineCurve, 1133
map
HxMatrix, 1000
HxVector, 1352
max
HxComplex, 516, 522
HxScalarDouble, 1155, 1159
HxScalarInt, 1173, 1178
HxVec2Double, 1272, 1277
HxVec2Int, 1291, 1295
HxVec3Double, 1311, 1316
HxVec3Int, 1331, 1336
maxAssign
HxComplex, 522
HxScalarDouble, 1159
HxScalarInt, 1178
HxVec2Double, 1277
HxVec2Int, 1295
HxVec3Double, 1316
HxVec3Int, 1336
maxBasis
HxBSplineBasis, 472
maxT
HxBSplineBasis, 474
HxBSplineCurve, 482
maxVal
HxHistogram, 568, 569
middle
HxBSplineInterval, 494
HxSampledBSplineInterval, 1144
min
HxComplex, 516, 521
HxScalarDouble, 1155, 1159
HxScalarInt, 1173, 1177
HxVec2Double, 1272, 1276
HxVec2Int, 1291, 1295
HxVec3Double, 1311, 1316
HxVec3Int, 1331, 1335
minAssign
HxComplex, 521
HxScalarDouble, 1159
HxScalarInt, 1177
HxVec2Double, 1276
HxVec2Int, 1295
HxVec3Double, 1316
HxVec3Int, 1336
minT

HxBSplineBasis, 473
HxBSplineCurve, 481
minVal
HxHistogram, 568
MIR_F
HxImageSeq, 648
mirrorBorder
HxImageData, 584
missed
VxStructureEval, 1364
MNPixOp
HxImageData, 594
HxImageList, 620
HxImageRep, 630
mod
HxComplex, 523
HxScalarDouble, 1161
HxScalarInt, 1179
HxVec2Double, 1278
HxVec2Int, 1297
HxVec3Double, 1317
HxVec3Int, 1337
modes
HxHistogram, 566
MPEG_F
HxImageSeq, 648
mul
HxMatrix, 998
HxVector, 1350
HxVectorR2, 1359
multiPixOp
HxImageData, 594
HxImageList, 619
HxImageRep, 630

name
HxImageData, 588
HxImageRep, 627
NameToSignature
HxImageSignature, 693
nCol
HxMatrix, 994
nearestKnot
HxBSplineBasis, 472
Neighbors
QThinning, 1360
neighbourhoodOp
HxImageData, 601, 603
HxImageRep, 636, 637
neighbourhoodOpExtra
HxImageData, 601
HxImageRep, 636
neighbourhoodOpExtra2
HxImageData, 602
HxImageRep, 636
nElem
HxMatrix, 994
HxVector, 1347
neutralElement
HxBpoAdd, 418
HxBpoAddAssign, 419
HxBpoAnd, 423
HxBpoDiv, 427
HxBpoHighlightRegion, 434
HxBpoInf, 436
HxBpoInfAssign, 438
HxBpoLeftShift, 439
HxBpoMax, 443
HxBpoMaxAssign, 445
HxBpoMin, 446
HxBpoMinAssign, 448
HxBpoMul, 451
HxBpoMulAssign, 452
HxBpoOr, 455
HxBpoRightShift, 458
HxBpoSub, 461
HxBpoSubAssign, 463
HxBpoSup, 466
HxBpoSupAssign, 468
next
HxBSplineInterval, 493
HxKerNgbNormCorrelation, 975
HxNgbBernsen, 1039
HxNgbDefuz, 1040
HxNgbHilditch, 1042
HxNgbIsMaxGradDir2d, 1047
HxNgbKuwahara, 1048
HxNgbLocalMode, 1051
HxNgbLWshed2d, 1055
HxNgbNonMaxSuppression2d, 1059
HxNgbPercentile2d, 1062
HxSampledBSplineInterval, 1144
next2
HxKerNgbNormCorrelation, 975
NextData
VideoReader, 1363
nImages
HxImageList, 617
nIntervals
HxBSplineBasis, 473
node
HxBSplineBasis, 472
norm1
HxComplex, 517
HxScalarDouble, 1155
HxScalarInt, 1174
HxVec2Double, 1273
HxVec2Int, 1291

HxVec3Double, 1311
HxVec3Int, 1331
norm2
HxComplex, 517
HxScalarDouble, 1155
HxScalarInt, 1174
HxVec2Double, 1273
HxVec2Int, 1291
HxVec3Double, 1312
HxVec3Int, 1331
normal
HxVectorR2, 1360
normalize
HxHistogram, 567
HxNJet, 1065
normInf
HxComplex, 517
HxScalarDouble, 1155
HxScalarInt, 1174
HxVec2Double, 1273
HxVec2Int, 1291
HxVec3Double, 1312
HxVec3Int, 1332
nrComponents
HxNJet, 1068
nrFrames
HxImageSeq, 651
HxImageSeqData, 656
HxImageSeqDXMedia, 660
HxImageSeqMDC, 668
nrOfBins
HxHistogram, 557
nRow
HxMatrix, 994
nrPhases
HxExportExtraIdentMaskCentralMoments, 533
nSamples
HxSampledBSplineCurve, 1134
nSources
HxMfMpo, 1025
numB
HxBSplineBasis, 474
numberOfPixels
HxImageData, 589
HxImageRep, 628
HxImageTem, 696
numKnots
HxLocalInterpol, 978
numP
HxBSplineCurve, 482
HxLocalInterpol, 978
HxSampledBSplineCurve, 1140
OC
HxAlternateSequentialFilter.h, 83
OCO
HxAlternateSequentialFilter.h, 83
operator ∗
HxComplex, 525
HxImageSeqIter, 664
HxMatrix, 1001, 1002, 1005
HxRcPtr, 1084
HxScalarDouble, 1163
HxScalarInt, 1181
HxVec2Double, 1280
HxVec2Int, 1299
HxVec2Tem, 1300
HxVec3Double, 1320
HxVec3Int, 1340
HxVec3Tem, 1341
HxVector, 1353
operator ∗=
HxComplex, 521
HxScalarDouble, 1158
HxScalarInt, 1177
HxVec2Double, 1276
HxVec2Int, 1294
HxVec2Tem, 1300
HxVec3Double, 1315
HxVec3Int, 1335
HxVec3Tem, 1341
operator double
HxVec2Tem, 1300
HxVec3Tem, 1341
operator HxComplex
HxScalarDouble, 1152
HxScalarInt, 1171
HxValue, 1261
HxVec2Double, 1270
HxVec2Int, 1288
HxVec3Double, 1308
HxVec3Int, 1328
operator HxImageSignature
HxIfRbPair, 581
operator HxScalarDouble
HxComplex, 513
HxScalarInt, 1170
HxValue, 1259
HxVec2Double, 1269
HxVec2Int, 1288
HxVec3Double, 1308
HxVec3Int, 1328
operator HxScalarInt
HxComplex, 513
HxScalarDouble, 1151
HxValue, 1259
HxVec2Double, 1269

HxVec2Int, 1287
HxVec3Double, 1308
HxVec3Int, 1328
operator HxString
HxClassName, 495
operator HxVec2Double
HxComplex, 513
HxScalarDouble, 1152
HxScalarInt, 1170
HxValue, 1260
HxVec2Int, 1288
HxVec2Tem, 1300
HxVec3Double, 1308
HxVec3Int, 1328
operator HxVec2Int
HxComplex, 513
HxScalarDouble, 1152
HxScalarInt, 1170
HxValue, 1259
HxVec2Double, 1269
HxVec2Tem, 1300
HxVec3Double, 1308
HxVec3Int, 1328
operator HxVec3Double
HxComplex, 513
HxScalarDouble, 1152
HxScalarInt, 1171
HxValue, 1260
HxVec2Double, 1270
HxVec2Int, 1288
HxVec3Int, 1328
HxVec3Tem, 1341
operator HxVec3Int
HxComplex, 513
HxScalarDouble, 1152
HxScalarInt, 1170
HxValue, 1260
HxVec2Double, 1269
HxVec2Int, 1288
HxVec3Double, 1308
HxVec3Tem, 1341
operator int
HxHistogram, 556
HxIfRbPair, 581
HxImageRep, 627
HxRcPtr, 1084
HxVec2Tem, 1300
HxVec3Tem, 1341
operator new
HxComplex, 511
HxScalarDouble, 1150
HxScalarInt, 1169
HxVec2Double, 1268
HxVec2Int, 1286
HxVec2Tem, 1300
HxVec3Double, 1306
HxVec3Int, 1326
HxVec3Tem, 1341
operator!=
HxCnum, 498
HxColor, 505
HxComplex, 514
HxImageSeqIter, 665
HxImageSignature, 690
HxImgFtorDescription, 704
HxMatrix, 1005
HxScalarDouble, 1153
HxScalarInt, 1171
HxVec2Double, 1270
HxVec2Int, 1289
HxVec3Double, 1309
HxVec3Int, 1329
HxVector, 1356
operator()
HxBlob2dPtrLess, 410
HxKernel1d, 971
HxKernel2d, 972
HxKernel3d, 972
operator+
HxComplex, 525
HxImageList, 618
HxMatrix, 1003
HxScalarDouble, 1162
HxScalarInt, 1181
HxVec2Double, 1280
HxVec2Int, 1299
HxVec2Tem, 1300
HxVec3Double, 1319
HxVec3Int, 1339
HxVec3Tem, 1341
HxVector, 1354
operator++
HxImageSeqIter, 663
operator+=
HxComplex, 520
HxImageList, 617
HxImageSeqIter, 664
HxScalarDouble, 1158
HxScalarInt, 1176
HxVec2Double, 1275
HxVec2Int, 1294
HxVec2Tem, 1300
HxVec3Double, 1315
HxVec3Int, 1335
HxVec3Tem, 1341
operator-
HxComplex, 515, 525
HxMatrix, 995, 1004

HxScalarDouble, 1153, 1163
HxScalarInt, 1172, 1181
HxVec2Double, 1271, 1280
HxVec2Int, 1289, 1299
HxVec2Tem, 1300
HxVec3Double, 1310, 1319
HxVec3Int, 1330, 1339
HxVec3Tem, 1341
HxVector, 1348, 1355
operator–
HxImageSeqIter, 664
operator-=
HxComplex, 520
HxScalarDouble, 1158
HxScalarInt, 1177
HxVec2Double, 1276
HxVec2Int, 1294
HxVec2Tem, 1300
HxVec3Double, 1315
HxVec3Int, 1335
HxVec3Tem, 1341
operator->
HxRcPtr, 1084
operator/
HxComplex, 525
HxMatrix, 1002, 1003
HxScalarDouble, 1163
HxScalarInt, 1181
HxVec2Double, 1280
HxVec2Int, 1299
HxVec2Tem, 1300
HxVec3Double, 1320
HxVec3Int, 1340
HxVec3Tem, 1341
HxVector, 1353, 1354
operator/=
HxComplex, 521
HxScalarDouble, 1159
HxScalarInt, 1177
HxVec2Double, 1276
HxVec2Int, 1295
HxVec2Tem, 1300
HxVec3Double, 1315
HxVec3Int, 1335
HxVec3Tem, 1341
operator<
HxComplex, 514
HxImageSignature, 690
HxImgFtorDescription, 704
HxPointAndValue, 1073
HxRegValue, 1106
HxScalarDouble, 1153
HxScalarInt, 1171
HxVec2Double, 1270
HxVec2Int, 1289
HxVec3Double, 1309
HxVec3Int, 1329
operator<<
HxColor.h, 107
HxGeoIntType.h, 231
HxPointList, 1074
HxPointZList, 1079
HxStringList, 1192
HxTagList.h, 365
HxValue, 1255
HxValueList, 1262
HxValueType.h, 381
operator<=
HxComplex, 514
HxScalarDouble, 1153
HxScalarInt, 1171
HxVec2Double, 1270
HxVec2Int, 1289
HxVec3Double, 1309
HxVec3Int, 1329
operator=
HxCnum, 497
HxColor, 500
HxComplex, 511
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 527
HxDataPtr3dScalarTem, 528
HxHistogram, 555
HxImageList, 617
HxImageRep, 627
HxImageSeq, 650
HxImageSeqIter, 663
HxImageSignature, 689
HxImgFtorDescription, 704
HxMatrix, 994
HxNJet, 1067
HxRcPtr, 1083
HxRegData, 1086
HxScalarDouble, 1150
HxSF, 1188
HxTagList, 1198
HxVec2Double, 1268
HxVector, 1347, 1348
operator==
HxColor, 505
HxComplex, 514
HxImageRep, 627
HxImageSeqIter, 665
HxImageSignature, 690
HxImgFtorDescription, 704
HxMatrix, 1005
HxScalarDouble, 1152
HxScalarInt, 1171

HxVec2Double, 1270
HxVec2Int, 1288
HxVec3Double, 1309
HxVec3Int, 1329
HxVector, 1356
operator>
HxComplex, 514
HxScalarDouble, 1153
HxScalarInt, 1172
HxVec2Double, 1271
HxVec2Int, 1289
HxVec3Double, 1309
HxVec3Int, 1329
operator>=
HxComplex, 514
HxScalarDouble, 1153
HxScalarInt, 1172
HxVec2Double, 1271
HxVec2Int, 1289
HxVec3Double, 1309
HxVec3Int, 1329
operator[]
HxImageList, 617
HxMatrix, 995
HxVector, 1348
opSeq
HxAlternateSequentialFilter.h, 83
or
HxComplex, 523
HxScalarDouble, 1161
HxScalarInt, 1179
HxVec2Double, 1278
HxVec2Int, 1297
HxVec3Double, 1318
HxVec3Int, 1338
ord2idx
HxNJet, 1071, 1072
order
HxNJet, 1067
origin
HxArrowR2, 401

P
HxBSplineCurve, 482
p
HxPointAndValue, 1073
part
HxBSplineInterval, 494
pathAffectedBy
HxBSplineBasis, 475
HxBSplineCurve, 483
PhaseCategory
HxExportExtraIdentMaskCentralMoments, 532
HxExportExtraIdentMaskMean, 535
HxExportExtraIdentMaskMedian, 537
HxExportExtraIdentMaskMoments, 540
HxExportExtraIdentMaskStdev, 542
HxExportExtraIdentMaskSum, 544
HxExportExtraWeightMaskSum, 546
HxKerNgbNormCorrelation, 974
HxNgbBernsen, 1039
HxNgbDefuz, 1041
HxNgbHilditch, 1043
HxNgbIsMaxGradDir2d, 1045
HxNgbKuwahara, 1048
HxNgbLocalMode, 1050
HxNgbLWshed2d, 1054
HxNgbNonMaxSuppression2d, 1057
HxNgbPercentile2d, 1061
pixelDimensionality
HxImageData, 589
HxImageRep, 628
HxImageSignature, 691
HxImageTem, 696
pixelPrecision
HxImageData, 589
HxImageRep, 629
HxImageSignature, 691
HxImageTem, 696
PixelType
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
pixelType
HxImageData, 589
HxImageRep, 628
HxImageSignature, 691
HxImageTem, 696
pointee

HxRcPtr, 1084
pointees
HxImageList, 618
pointer
HxPixelAllocator, 1072
PointValueT
QThinning, 1360
pow
HxComplex, 523
HxScalarDouble, 1160
HxScalarInt, 1179
HxVec2Double, 1278
HxVec2Int, 1296
HxVec3Double, 1317
HxVec3Int, 1337
preOpIsOk
HxMfBpo, 1009
HxMfExportExtra, 1012
HxMfGenConv, 1016
HxMfKernelNgb, 1020
HxMfMNpo, 1023
HxMfNgb, 1028
HxMfQueueBased, 1031
prev
HxBSplineInterval, 493
printInfo
HxImageData, 586
HxImageRep, 639
HxImageTem, 695
probeMNpo
HxImageData, 586
probeOp
HxImgFtorKernelNgb2d, 797
HxImgFtorMNpo, 805
HxImgFtorNgb2d, 812
HxImgFtorNgb2dExtra, 815
HxImgFtorNgb2dExtra2, 818
HxImgFunctor, 863
product
HxComplex, 516
HxScalarDouble, 1155
HxScalarInt, 1173
HxVec2Double, 1272
HxVec2Int, 1291
HxVec3Double, 1311
HxVec3Int, 1331
projectDomain
HxImageData, 586
HxImageTem2d, 697
HxImageTem3d, 699
ProjectDomainImageSigType
HxImageSig2dByte, 670
HxImageSig2dComplex, 670
HxImageSig2dDouble, 671
HxImageSig2dFloat, 672
HxImageSig2dInt, 673
HxImageSig2dShort, 673
HxImageSig2dVec2Byte, 674
HxImageSig2dVec2Double, 675
HxImageSig2dVec2Float, 676
HxImageSig2dVec2Int, 676
HxImageSig2dVec2Short, 677
HxImageSig2dVec3Byte, 678
HxImageSig2dVec3Double, 679
HxImageSig2dVec3Float, 679
HxImageSig2dVec3Int, 680
HxImageSig2dVec3Short, 681
HxImageSig3dByte, 682
HxImageSig3dDouble, 682
HxImageSig3dFloat, 683
HxImageSig3dInt, 684
HxImageSig3dShort, 685
projection
HxMatrix, 993
projectRange
HxImageData, 586
propagateBorder
HxImageData, 584
PThatAffectCAt
HxBSplineCurve, 483
PThatAffectSample
HxSampledBSplineCurve, 1136
put
HxArrowR2, 401
HxBlob2d, 406
HxBlob2dRelation, 413
HxBoundingBox, 417
HxColor, 505
HxComplex, 524
HxHistogram, 574
HxImageSeq, 653
HxImageSignature, 692
HxImgFtorDescription, 704
HxImgFtorKey, 802
HxImgFtorKeyNameTable, 803
HxImgFtorTable, 858
HxImgFunctor, 863
HxMatrix, 984
HxNameTable, 1038
HxNJet, 1071
HxPointR2, 1076
HxPolyline2d, 1082
HxRegData, 1088
HxRegistry, 1095
HxRegKey, 1103
HxRegValue, 1106
HxScalarDouble, 1162
HxScalarInt, 1180

HxSegmentation2d, 1186
HxTag, 1194
HxTagList, 1199
HxTagTem, 1203
HxVec2Double, 1279
HxVec2Int, 1298
HxVec2Tem, 1300
HxVec3Double, 1319
HxVec3Int, 1339
HxVec3Tem, 1341
HxVector, 1345
HxVectorR2, 1360

QT
QThinning, 1360
QThinning, 1360
~QThinning, 1360
calculate, 1361
className, 1361
first, 1361
getItemToQueue, 1361
getItemToRemove, 1361
getItemToWrite, 1361
globalPixelInit, 1360
killThisOne, 1361
localPixelInit, 1361
Neighbors, 1360
PointValueT, 1360
QT, 1360
QThinning, 1360
result3, 1361
VecNeighbors, 1360
wantAnotherLoop, 1361
wantFreshStartLocalPixelInit, 1360
QueryInterface
VideoReader, 1363
QueryResultType
HxImgFtorRuleBase, 842
queueBasedOp
HxImageData, 603
HxImageRep, 637
QueueT
HxImgFtorQueueBased, 823
queuet
HxImgFtorQueueBased, 823

ratio
HxBSplineInterval, 494
HxSampledBSplineInterval, 1144
read
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529
readIncX
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529
recGenConv
HxImageData, 599
HxImageRep, 635
recGenConv2dSep
HxImageData, 600
HxImageRep, 635
reduceOp
HxImageRep, 632
reduceRange
HxHistogram, 577
reduceRangeVal
HxHistogram, 578
ref
HxImageRep, 625
refCnt
HxRcObject, 1083
HxRcPtr, 1084
reference
HxPixelAllocator, 1072
reflect2d
HxMatrix, 990
reflect3d
HxMatrix, 992
removeRef
HxRcObject, 1083
render3d
HxHistogram, 573
resample
HxNJet, 1065
restrict
HxImageData, 586
result
HxKerNgbNormCorrelation, 976
HxMfBpo, 1009
HxMfDiy, 1010
HxMfGenConv, 1016
HxMfIdentity, 1018
HxMfKernelNgb, 1020
HxMfMpo, 1026
HxMfNgb, 1028
HxMfQueueBased, 1031
HxMfResize, 1033
HxMfUpo, 1035
HxNgbBernsen, 1039
HxNgbDefuz, 1040
HxNgbHilditch, 1042
HxNgbIsMaxGradDir2d, 1047
HxNgbKuwahara, 1048
HxNgbLocalMode, 1051
HxNgbLWshed2d, 1055
HxNgbNonMaxSuppression2d, 1059

HxNgbPercentile2d, 1063
result3
QThinning, 1361
resultCnt
HxMfMNpo, 1023
ResultPrecision
HxImageRep, 626
results
HxMfMNpo, 1023
RGB2Intensity, 1361
className, 1362
doIt, 1362
RGB2Intensity, 1362
TransVarianceCategory, 1362
rgbOp
HxImageData, 604
rightShift
HxComplex, 524
HxScalarDouble, 1161
HxScalarInt, 1180
HxVec2Double, 1279
HxVec2Int, 1298
HxVec3Double, 1318
HxVec3Int, 1338
rotate
HxNJet, 1065
rotate2d
HxMatrix, 989
rotate2dDeg
HxMatrix, 989
rotateDeg
HxNJet, 1065
rotateSF
HxSF, 1189
rotateX3d
HxMatrix, 991
rotateX3dDeg
HxMatrix, 991
rotateY3d
HxMatrix, 991
rotateY3dDeg
HxMatrix, 992
rotateZ3d
HxMatrix, 992
rotateZ3dDeg
HxMatrix, 992
round
HxComplex, 516
HxScalarDouble, 1154
HxScalarInt, 1173
HxVec2Double, 1272
HxVec2Int, 1290
HxVec3Double, 1310
HxVec3Int, 1330
RuleType
HxImgFtorBpo, 701
RvType
HxRegData, 1085

sampleC
HxBSplineCurve, 486
sampledInterval
HxSampledBSplineCurve, 1135
sampledT
HxSampledBSplineCurve, 1134
samplesAffectedBy
HxSampledBSplineCurve, 1136
samplingAlg
HxSampledBSplineCurve, 1134
scale
HxNJet, 1068
scale2d
HxMatrix, 989
scale3d
HxMatrix, 990
scaleAllP
HxBSplineCurve, 487
set
HxImageData, 584
HxImageTem, 694
setArguments
HxImgFtorKey, 801
setArgumentType
HxImgFtorRuleBase, 844
setAt
HxImageData, 586
HxImageRep, 639
HxImageTem, 695
setBin
HxHistogram, 564, 565
setBorder
HxImageData, 584, 592
setCursorKey
HxRegistry, 1094
setCursorUp
HxRegistry, 1094
setData
HxRegValue, 1106
SetDataSizes
VideoReader, 1363
setDefaultResultPrecision
HxImageRep, 632
setExtra2Type
HxImgFtorRuleBase, 845
setExtraType
HxImgFtorRuleBase, 845
setImageDataObserver
HxImageRep, 625

setImageDimensionality
HxImageSignature, 691
setInputImage
HxSegmentation2d, 1184
setInt
HxRegData, 1087
setIsModifying
HxImgFtorRuleBase, 846
setKernelType
HxImgFtorRuleBase, 844
setKey
HxImgFtorDescription, 704
setLabeledImage
HxSegmentation2d, 1184
setObjectObserver
HxImageData, 586
HxImageRep, 625
setPartImage
HxImageData, 584, 592
setPixelDimensionality
HxImageSignature, 691
setPixelPrecision
HxImageSignature, 692
setPixelType
HxImageSignature, 692
setPpmPixels
HxImageData, 586
setResultType
HxImgFtorRuleBase, 843
setRootKey
HxRegistry, 1095
setString
HxRegData, 1087
setTags
HxImgFtorDescription, 704
setUseMDC
HxImageSeq, 650
setValue
HxComplex, 507
HxScalarDouble, 1146
HxScalarInt, 1164
HxVec2Double, 1263
HxVec2Int, 1282
HxVec3Double, 1301
HxVec3Int, 1321
setVariation
HxImgFtorDescription, 704
sgn
HxMatrix, 1000
HxVector, 1352
shear2d
HxMatrix, 990
sig
HxIfRbPair, 581
signature
HxImageData, 589
HxImageRep, 629
HxImageTem, 696
sin
HxComplex, 517
HxMatrix, 999
HxScalarDouble, 1156
HxScalarInt, 1174
HxVec2Double, 1273
HxVec2Int, 1292
HxVec3Double, 1312
HxVec3Int, 1332
HxVector, 1350
sinh
HxComplex, 519
HxMatrix, 999
HxScalarDouble, 1157
HxScalarInt, 1175
HxVec2Double, 1274
HxVec2Int, 1293
HxVec3Double, 1314
HxVec3Int, 1333
HxVector, 1351
size
HxBoundingBox, 416
HxImageList, 617
HxKerNgbNormCorrelation, 975
HxNgbBernsen, 1039
HxNgbDefuz, 1040
HxNgbHilditch, 1042
HxNgbIsMaxGradDir2d, 1046
HxNgbKuwahara, 1048
HxNgbLocalMode, 1051
HxNgbLWshed2d, 1054
HxNgbNonMaxSuppression2d, 1058
HxNgbPercentile2d, 1062
HxSampledBSplineInterval, 1145
size_type
HxPixelAllocator, 1072
sizeMaer
HxBlob2d, 404
sizes
HxImageData, 588
HxImageRep, 628
HxImageTem, 695
HxKernel1d, 971
HxKernel2d, 972
HxKernel3d, 972
SizeType
HxImgFtorKeyNameTable, 803
sizeType
HxNameTable, 1036
SMALL_VAL

HxComplex, 526
HxScalarDouble, 1163
HxScalarInt, 1181
HxVec2Double, 1281
HxVec2Int, 1299
HxVec3Double, 1320
HxVec3Int, 1340
smooth
HxHistogram, 566
source
HxMfDiy, 1010
HxMfExportExtra, 1012
HxMfGenConv, 1015
HxMfIdentity, 1017
HxMfKernelNgb, 1020
HxMfNgb, 1028
HxMfQueueBased, 1031
HxMfResize, 1033
HxMfUpo, 1035
source1
HxMfBpo, 1008
source2
HxMfBpo, 1009
sourceCnt
HxMfMNpo, 1023
sources
HxMfMNpo, 1023
HxMfMpo, 1025
splitString
HxStringList.h, 357
sqrt
HxComplex, 517
HxMatrix, 1000
HxScalarDouble, 1156
HxScalarInt, 1174
HxVec2Double, 1273
HxVec2Int, 1292
HxVec3Double, 1312
HxVec3Int, 1332
HxVector, 1352
squaredMagnitude
HxVectorR2, 1359
SrcDataPtrArray
HxImgFtorIMCast, 781
HxImgFtorIMNCast, 788
SrcDataPtrType
HxImgFtorIMCast, 781
HxImgFtorIMNCast, 788
startMaer
HxBlob2d, 404
STDMETHODIMP_
VideoReader, 1363
sub
HxMatrix, 997, 998
HxPointR2, 1076
HxVector, 1349
HxVectorR2, 1358
subscribeGenerator
HxImageFactory, 606
sum
HxComplex, 516
HxHistogram, 567
HxScalarDouble, 1154
HxScalarInt, 1173
HxVec2Double, 1272
HxVec2Int, 1290
HxVec3Double, 1311
HxVec3Int, 1331
sup
HxComplex, 522
HxScalarDouble, 1160
HxScalarInt, 1178
HxVec2Double, 1277
HxVec2Int, 1296
HxVec3Double, 1317
HxVec3Int, 1337
supAssign
HxComplex, 522
HxScalarDouble, 1160
HxScalarInt, 1179
HxVec2Double, 1278
HxVec2Int, 1296
HxVec3Double, 1317
HxVec3Int, 1337
svd
HxMatrix, 996

t
HxMatrix, 996
HxVector, 1348
Tag
HxValue, 1255
tag
HxValue, 1257
TagPtr
HxTagList, 1197
tan
HxComplex, 518
HxMatrix, 999
HxScalarDouble, 1156
HxScalarInt, 1175
HxVec2Double, 1274
HxVec2Int, 1292
HxVec3Double, 1313
HxVec3Int, 1332
HxVector, 1351
tanh
HxComplex, 519

HxMatrix, 999
HxScalarDouble, 1157
HxScalarInt, 1176
HxVec2Double, 1275
HxVec2Int, 1293
HxVec3Double, 1314
HxVec3Int, 1334
HxVector, 1351
threshold
HxHistogram, 576
to1D
HxHistogram, 579
toCMY
HxColor, 501
toFile
HxNJet, 1067
toHSI
HxColor, 504
toLab
HxColor, 502
toLuv
HxColor, 503
toOOO
HxColor, 503
toRGB
HxColor, 501
toString
HxColor, 505
HxComplex, 524
HxImageSignature, 692
HxImgFtorDescription, 704
HxImgFtorKey, 802
HxPointR2, 1075
HxRegData, 1088
HxScalarDouble, 1162
HxScalarInt, 1180
HxTag1Phase, 1194
HxTag2Phase, 1195
HxTagCnum, 1196
HxTagList, 1199
HxTagLoop, 1200
HxTagNPhase, 1200
HxTagPixOpIn, 1201
HxTagPixOpOut, 1202
HxTagTransInVar, 1204
HxTagTransVar, 1205
HxValue, 1255
HxVec2Double, 1279
HxVec2Int, 1298
HxVec3Double, 1319
HxVec3Int, 1339
HxVectorR2, 1357
toXYZ
HxColor, 502
translate
HxBoundingBox, 417
translate2d
HxMatrix, 989
translate3d
HxMatrix, 990
translateAllP
HxBSplineCurve, 487
translateCurve
HxBSplineCurve, 487, 488
HxSampledBSplineCurve, 1141
translateCurve2
HxBSplineCurve, 488
TransVarianceCategory
HxBpoAdd, 418
HxBpoAddAssign, 420
HxBpoAddSat, 421
HxBpoAnd, 423
HxBpoBind2Val, 424
HxBpoCross, 426
HxBpoDiv, 427
HxBpoDot, 429
HxBpoEqual, 430
HxBpoGreaterEqual, 432
HxBpoGreaterThan, 433
HxBpoHighlightRegion, 435
HxBpoInf, 436
HxBpoInfAssign, 438
HxBpoLeftShift, 439
HxBpoLessEqual, 441
HxBpoLessThan, 442
HxBpoMax, 444
HxBpoMaxAssign, 445
HxBpoMin, 447
HxBpoMinAssign, 448
HxBpoMod, 450
HxBpoMul, 451
HxBpoMulAssign, 453
HxBpoNotEqual, 454
HxBpoOr, 456
HxBpoPow, 457
HxBpoRightShift, 459
HxBpoSqrDst, 460
HxBpoSub, 462
HxBpoSubAssign, 463
HxBpoSubSat, 465
HxBpoSup, 467
HxBpoSupAssign, 468
HxBpoXor, 470
HxExportExtraIdentMaskCentralMoments, 532
HxExportExtraIdentMaskMean, 535
HxExportExtraIdentMaskMedian, 537
HxExportExtraIdentMaskMoments, 540

HxExportExtraIdentMaskStdev, 542
HxExportExtraIdentMaskSum, 544
HxExportExtraWeightMaskSum, 546
HxUpoAbs, 1206
HxUpoAcos, 1207
HxUpoArg, 1209
HxUpoAsin, 1210
HxUpoAtan, 1212
HxUpoAtan2, 1213
HxUpoCeil, 1215
HxUpoColSpace, 1216
HxUpoComplement, 1218
HxUpoConjugate, 1219
HxUpoCos, 1221
HxUpoCosh, 1222
HxUpoExp, 1224
HxUpoFloor, 1225
HxUpoLog, 1227
HxUpoLog10, 1228
HxUpoMax, 1230
HxUpoMin, 1231
HxUpoNegate, 1233
HxUpoNorm1, 1234
HxUpoNorm2, 1236
HxUpoNorm2Sqr, 1237
HxUpoNormInf, 1239
HxUpoProduct, 1240
HxUpoRound, 1242
HxUpoSin, 1243
HxUpoSinh, 1245
HxUpoSqrt, 1246
HxUpoSum, 1248
HxUpoTan, 1249
HxUpoTanh, 1251
HxUpoTriStateThreshold, 1252
RGB2Intensity, 1362

truncate
HxNJet, 1065

type
HxRegData, 1087

unaryPixOp
HxImageData, 593
HxImageList, 618
HxImageRep, 629

unite
HxBoundingBox, 416

unSubscribeGenerator
HxImageFactory, 606

v
HxPointAndValue, 1073

valid
HxImageSeqData, 655
HxImageSeqDXMedia, 659
HxImageSeqMDC, 667
HxMatrix, 994
HxVector, 1347

value
HxColor, 500
HxVec2Tem, 1300
HxVec3Tem, 1341

value_type
HxPixelAllocator, 1072

valueExists
HxRegistry, 1094

valueListSize
HxRegKey, 1103

valueToBin
HxHistogram, 558

VecNeighbors
QThinning, 1360

VecNeighborT
HxImgFtorQueueBased, 823

VecPointT
HxImgFtorQueueBased, 823

VideoReader
~VideoReader, 1363
getFrame, 1363
getFrameHeight, 1363
getFrameWidth, 1363
getLength, 1363
isNull, 1363
NextData, 1363
QueryInterface, 1363
SetDataSizes, 1363
STDMETHODIMP_, 1363
VideoReader, 1363

VideoReader, 1363

vprint
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529

VxStructureEval
correct, 1364
falseAlarm, 1364
missed, 1364

VxStructureEval, 1363

wantAnotherLoop
QThinning, 1361

wantFreshStartLocalPixelInit
QThinning, 1360

weight
HxImageData, 586

width
HxImageTem2d, 697
HxImageTem3d, 699

write
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529
HxHistogram, 575
writeFile
HxImageFactory, 613
HxImageSeq, 652
writeIncX
HxDataPtr2dScalarTem, 527
HxDataPtr2dTem, 528
HxDataPtr3dScalarTem, 529

x
HxCnum, 497
HxComplex, 512
HxCoord, 526
HxPointR2, 1076
HxScalarDouble, 1151
HxScalarInt, 1170
HxVec2Double, 1268
HxVec2Int, 1287
HxVec2Tem, 1300
HxVec3Double, 1307
HxVec3Int, 1327
HxVec3Tem, 1341
HxVectorR2, 1358
xor
HxComplex, 523
HxScalarDouble, 1161
HxScalarInt, 1179
HxVec2Double, 1278
HxVec2Int, 1297
HxVec3Double, 1318
HxVec3Int, 1338
xy
HxNJet, 1068
xyl
HxNJet, 1068
xyz
HxNJet, 1068
xyzl
HxNJet, 1069

y
HxCnum, 497
HxComplex, 512
HxCoord, 526
HxPointR2, 1076
HxVec2Double, 1269
HxVec2Int, 1287
HxVec2Tem, 1300
HxVec3Double, 1307
HxVec3Int, 1327
HxVec3Tem, 1341
HxVectorR2, 1358

z
HxCnum, 498
HxCoord, 526
HxVec3Double, 1307
HxVec3Int, 1327
HxVec3Tem, 1341