This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

This software and related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

Copyright © 1993, 2012, Oracle and/or its affiliates. All rights reserved.
## Contents

**Preface** ...................................................................................................................................................31

**Multimedia Library Functions - Part 1** ....................................................................................................35
  mlib_free(3MLIB) ..................................................................................................................................36
  mlib_GraphicsBoundaryFill_8(3MLIB) .................................................................................................37
  mlib_GraphicsDrawArc(3MLIB) ............................................................................................................38
  mlib_GraphicsDrawCircle(3MLIB) .......................................................................................................40
  mlib_GraphicsDrawEllipse(3MLIB) ......................................................................................................42
  mlib_GraphicsDrawLine(3MLIB) ..........................................................................................................44
  mlib_GraphicsDrawLineFanSet(3MLIB) .................................................................................................49
  mlib_GraphicsDrawLineSet(3MLIB) ....................................................................................................54
  mlib_GraphicsDrawLineStripSet(3MLIB) .............................................................................................59
  mlib_GraphicsDrawPoint(3MLIB) .........................................................................................................64
  mlib_GraphicsDrawPointSet(3MLIB) ..................................................................................................66
  mlib_GraphicsDrawPolygon(3MLIB) ....................................................................................................68
  mlib_GraphicsDrawPolyline(3MLIB) ....................................................................................................73
  mlib_GraphicsDrawRectangle(3MLIB) .................................................................................................78
  mlib_GraphicsDrawTriangle(3MLIB) ...................................................................................................80
  mlib_GraphicsDrawTriangleFanSet(3MLIB) ..........................................................................................85
  mlib_GraphicsDrawTriangleSet(3MLIB) ...............................................................................................90
  mlib_GraphicsDrawTriangleStripSet(3MLIB) ....................................................................................95
  mlib_GraphicsFillArc(3MLIB) ............................................................................................................100
  mlib_GraphicsFillCircle(3MLIB) .......................................................................................................102
  mlib_GraphicsFillEllipse(3MLIB) .....................................................................................................104
  mlib_GraphicsFillPolygon(3MLIB) ....................................................................................................106
Contents

Multimedia Library Functions - Part 2

mlib_GraphicsFillRectangle(3MLIB) ................................................................. 111
mlib_GraphicsFillTriangle(3MLIB) ............................................................... 112
mlib_GraphicsFillTriangleFanSet(3MLIB) .................................................... 114
mlib_GraphicsFillTriangleSet(3MLIB) ......................................................... 119
mlib_GraphicsFillTriangleStripSet(3MLIB) .............................................. 124
mlib_GraphicsFloodFill_8(3MLIB) ............................................................. 129
mlib_GraphicsFloodFill_8_Inp(3MLIB) ....................................................... 134
mlib_ImageAbs(3MLIB) ........................................................................ 135
mlib_ImageAbs_Fp(3MLIB) ..................................................................... 136
mlib_ImageAbs_Fp_Inp(3MLIB) ................................................................. 137
mlib_ImageAbs_Inp(3MLIB) .................................................................... 138
mlib_ImageAdd(3MLIB) ........................................................................ 139
mlib_ImageAdd_Fp(3MLIB) ..................................................................... 140
mlib_ImageAdd_Fp_Inp(3MLIB) ............................................................... 141
mlib_ImageAdd_Inp(3MLIB) .................................................................... 142
mlib_ImageAffine(3MLIB) ...................................................................... 143
mlib_ImageAffine_Fp(3MLIB) .................................................................. 145
mlib_ImageAffineIndex(3MLIB) ................................................................. 147
mlib_ImageAffineTable(3MLIB) ................................................................. 149
mlib_ImageAffineTable_Fp(3MLIB) .......................................................... 151
mlib_ImageAffineTransform(3MLIB) ........................................................ 153
mlib_ImageAffineTransform_Fp(3MLIB) ................................................... 155
mlib_ImageAffineTransformIndex(3MLIB) ............................................... 157
mlib_ImageAnd(3MLIB) .......................................................................... 159
mlib_ImageAnd_Inp(3MLIB) .................................................................... 160
mlib_ImageAndNot1_Inp(3MLIB) ............................................................. 161
mlib_ImageAndNot2_Inp(3MLIB) ............................................................. 162
mlib_ImageAndNot(3MLIB) .................................................................... 163
mlib_ImageAutoCorrel(3MLIB) ................................................................. 164
mlib_ImageAutoCorrel_Fp(3MLIB) ............................................................ 165
mlib_ImageAve(3MLIB) .......................................................................... 166
mlib_ImageAve_Fp(3MLIB) ..................................................................... 167
mlib_ImageAve_Fp_Inp(3MLIB) ............................................................... 168
mlib_ImageAve_Inp(3MLIB) .................................................................... 169
mlib_ImageBlend1_Fp_Inp(3MLIB) ........................................................... 170
mlib_ImageBlend1_Inp(3MLIB) ............................................................... 171
mlib_ImageConstMul(3MLIB) ........................................................................................................270
mlib_ImageConstMul_Fp(3MLIB) ..................................................................................................271
mlib_ImageConstMul_Fp_Inp(3MLIB) ..........................................................................................272
mlib_ImageConstMul_Inp(3MLIB) .............................................................................................273
mlib_ImageConstMulShift(3MLIB) ...............................................................................................274
mlib_ImageConstMulShift_Inp(3MLIB) ........................................................................................275
mlib_ImageConstNotAnd(3MLIB) .................................................................................................276
mlib_ImageConstNotAnd_Inp(3MLIB) ..........................................................................................277
mlib_ImageConstNotOr(3MLIB) ....................................................................................................278
mlib_ImageConstNotOr_Inp(3MLIB) .............................................................................................279
mlib_ImageConstNotXor(3MLIB) ..................................................................................................280
mlib_ImageConstNotXor_Inp(3MLIB) ..........................................................................................281
mlib_ImageConstOr(3MLIB) ..........................................................................................................282
mlib_ImageConstOr_Inp(3MLIB) ..................................................................................................283
mlib_ImageConstOrNot(3MLIB) ...................................................................................................284
mlib_ImageConstOrNot_Inp(3MLIB) .............................................................................................285
mlib_ImageConstSub(3MLIB) .......................................................................................................286
mlib_ImageConstSub_Fp(3MLIB) ..................................................................................................287
mlib_ImageConstSub_Fp_Inp(3MLIB) ..........................................................................................288
mlib_ImageConstSub_Inp(3MLIB) ...............................................................................................289
mlib_ImageConstXor(3MLIB) .......................................................................................................290
mlib_ImageConstXor_Inp(3MLIB) ................................................................................................291
mlib_ImageConv2x2(3MLIB) .........................................................................................................292
mlib_ImageConv2x2_Fp(3MLIB) ....................................................................................................294
mlib_ImageConv2x2Index(3MLIB) ...............................................................................................296
mlib_ImageConv3x3(3MLIB) .........................................................................................................298
mlib_ImageConv3x3_Fp(3MLIB) ....................................................................................................300
mlib_ImageConv3x3Index(3MLIB) ...............................................................................................302
mlib_ImageConv4x4(3MLIB) .........................................................................................................304
mlib_ImageConv4x4_Fp(3MLIB) ....................................................................................................306
mlib_ImageConv4x4Index(3MLIB) ...............................................................................................308
mlib_ImageConv5x5(3MLIB) .........................................................................................................310
mlib_ImageConv5x5_Fp(3MLIB) ..................................................................................................312
mlib_ImageConv5x5Index(3MLIB) ...............................................................................................314
mlib_ImageConv7x7(3MLIB) .........................................................................................................316
mlib_ImageConv7x7_Fp(3MLIB) ..................................................................................................318
Multimedia Library Functions - Part 3

mlib_ImageDivShift2_Inp(3MLIB) ........................................................................... 365
mlib_ImageDivShift(3MLIB) .................................................................................. 366
mlib_ImageErode4(3MLIB) ..................................................................................... 367
mlib_ImageErode4_Fp(3MLIB) .................................................................................. 368
mlib_ImageErode8(3MLIB) ..................................................................................... 369
mlib_ImageErode8_Fp(3MLIB) ................................................................................ 370
mlib_ImageExp(3MLIB) .......................................................................................... 371
mlib_ImageExp_Fp(3MLIB) ...................................................................................... 372
mlib_ImageExp_Fp_Inp(3MLIB) ................................................................................ 373
mlib_ImageExp_Inp(3MLIB) ..................................................................................... 374
mlib_ImageExtrema2(3MLIB) .................................................................................. 375
mlib_ImageExtremaLocations(3MLIB) .................................................................... 376
mlib_ImageFilteredSubsample(3MLIB) .................................................................. 377
mlib_ImageFlipAntiDiag(3MLIB) ........................................................................... 378
mlib_ImageFlipAntiDiag_Fp(3MLIB) ...................................................................... 379
mlib_ImageFlipMainDiag(3MLIB) ......................................................................... 380
mlib_ImageFlipMainDiag_Fp(3MLIB) ..................................................................... 381
mlib_ImageFlipX(3MLIB) ...................................................................................... 382
mlib_ImageFlipX_Fp(3MLIB) ................................................................................ 383
mlib_ImageFlipY(3MLIB) ...................................................................................... 384
mlib_ImageFlipY_Fp(3MLIB) ................................................................................ 385
mlib_ImageFourierTransform(3MLIB) ................................................................. 386
mlib_ImageGetBitOffset(3MLIB) ......................................................................... 387
mlib_ImageGetChannels(3MLIB) ......................................................................... 388
mlib_ImageGetData(3MLIB) .................................................................................. 389
mlib_ImageGetFlags(3MLIB) ................................................................................ 390
mlib_ImageGetFormat(3MLIB) .............................................................................. 391
mlib_ImageGetHeight(3MLIB) .............................................................................. 392
mlib_ImageGetPaddings(3MLIB) ......................................................................... 393
mlib_ImageGetStride(3MLIB) .............................................................................. 394
mlib_ImageGetType(3MLIB) ................................................................................ 395
mlib_ImageGetWidth(3MLIB) .............................................................................. 396
mlib_ImageGradient3x3(3MLIB) ........................................................................... 397
mlib_ImageGradient3x3_Fp(3MLIB) ...................................................................... 398
mlib_ImageGradientMxN(3MLIB) .......................................................................... 399
mlib_ImageGradientMxN_Fp(3MLIB) ..................................................................... 400
mlib_ImageGradientMxN_Fp_Inp(3MLIB) ................................................................. 401
mlib_ImageTransform(3MLIB) ............................................................................. 402
mlib_ImageTransform_Fp(3MLIB) ........................................................................ 403
mlib_ImageTransform_Fp_Inp(3MLIB) ................................................................ 404
mlib_ImageTransform_Inp(3MLIB) ...................................................................... 405
mlib_ImageVariance(3MLIB) ............................................................................... 406
mlib_ImageVariance_Fp(3MLIB) .......................................................................... 407
mlib_ImageVariance_Inp(3MLIB) ........................................................................... 408
mlib_ImageMoment2(3MLIB) .......................................................................................... 520
mlib_ImageMoment2_Fp(3MLIB) .......................................................................................... 521
mlib_ImageMulAlpha(3MLIB) ............................................................................................ 522
mlib_ImageMulAlpha_Fp(3MLIB) .......................................................................................... 524
mlib_ImageMulAlpha_Fp_Inp(3MLIB) .................................................................................. 525
mlib_ImageMulAlpha_Inp(3MLIB) ...................................................................................... 526
mlib_ImageMul_Fp(3MLIB) .................................................................................................. 527
mlib_ImageMul_Fp_Inp(3MLIB) ............................................................................................ 528
mlib_ImageMulShift(3MLIB) .............................................................................................. 529
mlib_ImageMulShift_Inp(3MLIB) ....................................................................................... 530
mlib_ImageNormCrossCorrel(3MLIB) ................................................................................ 531
mlib_ImageNormCrossCorrel_Fp(3MLIB) ............................................................................ 534
mlib_ImageNot(3MLIB) ...................................................................................................... 537
mlib_ImageNotAnd(3MLIB) ............................................................................................... 538
mlib_ImageNotAnd_Inp(3MLIB) .......................................................................................... 539
mlib_ImageNot_Inp(3MLIB) ................................................................................................ 540
mlib_ImageNotOr(3MLIB) ................................................................................................... 541
mlib_ImageNotOr_Inp(3MLIB) ............................................................................................ 542
mlib_ImageNotXor(3MLIB) ............................................................................................... 543
mlib_ImageNotXor_Inp(3MLIB) .......................................................................................... 544
mlib_ImageOr(3MLIB) ........................................................................................................ 545
mlib_ImageOr_Inp(3MLIB) .................................................................................................. 546
mlib_ImageOrNot1_Inp(3MLIB) .......................................................................................... 547
mlib_ImageOrNot2_Inp(3MLIB) .......................................................................................... 548
mlib_ImageOrNot(3MLIB) .................................................................................................. 549
mlib_ImagePolynomialWarp(3MLIB) ................................................................................ 550
mlib_ImagePolynomialWarp_Fp(3MLIB) ............................................................................ 553
mlib_ImagePolynomialWarpTable(3MLIB) ....................................................................... 556
mlib_ImagePolynomialWarpTable_Fp(3MLIB) ................................................................ 559
mlib_ImageRankFilter3x3(3MLIB) .................................................................................... 562
mlib_ImageRankFilter3x3_Fp(3MLIB) ................................................................................ 564
mlib_ImageRankFilter3x3_US(3MLIB) ............................................................................... 566
mlib_ImageRankFilter5x5(3MLIB) ..................................................................................... 568
mlib_ImageRankFilter5x5_Fp(3MLIB) ............................................................................... 570
mlib_ImageRankFilter5x5_US(3MLIB) ............................................................................... 572
mlib_ImageRankFilter7x7(3MLIB) .................................................................................... 574
mlib_ImageSConv7x7_Fp(3MLIB) ........................................................................................................... 632
mlib_ImageSConvKernelConvert(3MLIB) ................................................................................................. 634
mlib_ImageSetFormat(3MLIB) ................................................................................................................ 635
mlib_ImageSetPaddings(3MLIB) ............................................................................................................. 636
mlib_ImageSetStruct(3MLIB) .................................................................................................................. 638
mlib_ImageSetSubimageStruct(3MLIB) ................................................................................................... 640
mlib_ImageSobel(3MLIB) ....................................................................................................................... 642
mlib_ImageSqr_Fp(3MLIB) ...................................................................................................................... 644
mlib_ImageSqr_Fp_Inp(3MLIB) ............................................................................................................. 645
mlib_ImageSqrShift(3MLIB) .................................................................................................................. 646
mlib_ImageSqrShift_Inp(3MLIB) ............................................................................................................ 647
mlib_ImageStdDev(3MLIB) ..................................................................................................................... 648
mlib_ImageStdDev_Fp(3MLIB) ................................................................................................................ 649
mlib_ImageSub1_Fp_Inp(3MLIB) .............................................................................................................. 650
mlib_ImageSub1_Inp(3MLIB) .................................................................................................................. 651
mlib_ImageSub2_Fp_Inp(3MLIB) .............................................................................................................. 652
mlib_ImageSub2_Inp(3MLIB) .................................................................................................................. 653
mlib_ImageSub(3MLIB) ........................................................................................................................... 654
mlib_ImageSub_Fp(3MLIB) ...................................................................................................................... 655
mlib_ImageSubsampleAverage(3MLIB) ................................................................................................. 656
mlib_ImageSubsampleBinaryToGray(3MLIB) ......................................................................................... 658
mlib_ImageTestFlags(3MLIB) ................................................................................................................ 660
mlib_ImageThresh1(3MLIB) .................................................................................................................... 661
mlib_ImageThresh1_Fp(3MLIB) ................................................................................................................. 663
mlib_ImageThresh1_Fp_Inp(3MLIB) ......................................................................................................... 665
mlib_ImageThresh1_Inp(3MLIB) ................................................................................................................. 666
mlib_ImageThresh2(3MLIB) .................................................................................................................... 667
mlib_ImageThresh2_Fp(3MLIB) ................................................................................................................. 668
mlib_ImageThresh2_Fp_Inp(3MLIB) ......................................................................................................... 669
mlib_ImageThresh2_Inp(3MLIB) ................................................................................................................. 670
mlib_ImageThresh3(3MLIB) .................................................................................................................... 671
mlib_ImageThresh3_Fp(3MLIB) ................................................................................................................. 672
mlib_ImageThresh3_Fp_Inp(3MLIB) ......................................................................................................... 673
mlib_ImageThresh3_Inp(3MLIB) ................................................................................................................. 674
mlib_ImageThresh4(3MLIB) .................................................................................................................... 675
mlib_ImageThresh4_Fp(3MLIB) ................................................................................................................. 677
mlib_ImageThresh4_Fp_Inp(3MLIB) ................................................................. 679
mlib_ImageThresh4_Inp(3MLIB) ........................................................................... 681
mlib_ImageThresh5(3MLIB) ................................................................................... 683
mlib_ImageThresh5_Fp(3MLIB) ............................................................................. 684
mlib_ImageThresh5_Fp_Inp(3MLIB) ........................................................................ 685
mlib_ImageThresh5_Inp(3MLIB) ............................................................................ 686
mlib_ImageXor(3MLIB) .......................................................................................... 687
mlib_ImageXor_Inp(3MLIB) ................................................................................... 688
mlib_ImageXProj(3MLIB) ...................................................................................... 689
mlib_ImageXProj_Fp(3MLIB) ................................................................................... 690
mlib_ImageYProj(3MLIB) ...................................................................................... 691
mlib_ImageYProj_Fp(3MLIB) ................................................................................... 692
mlib_ImageZoom(3MLIB) ...................................................................................... 693
mlib_ImageZoomBlend(3MLIB) ............................................................................. 695
mlib_ImageZoom_Fp(3MLIB) .................................................................................. 697
mlib_ImageZoomIn2X(3MLIB) .............................................................................. 699
mlib_ImageZoomIn2X_Fp(3MLIB) ......................................................................... 700
mlib_ImageZoomIn2XIndex(3MLIB) ...................................................................... 701
mlib_ImageZoomIndex(3MLIB) ............................................................................. 702
mlib_ImageZoomOut2X(3MLIB) .......................................................................... 703
mlib_ImageZoomOut2X_Fp(3MLIB) ...................................................................... 705
mlib_ImageZoomOut2XIndex(3MLIB) ................................................................... 706
mlib_ImageZoomTranslate(3MLIB) ..................................................................... 707
mlib_ImageZoomTranslateBlend(3MLIB) ............................................................ 709
mlib_ImageZoomTranslate_Fp(3MLIB) .................................................................. 710
mlib_ImageZoomTranslateTable(3MLIB) ............................................................ 711
mlib_ImageZoomTranslateTableBlend(3MLIB) .................................................... 712
mlib_ImageZoomTranslateTable_Fp(3MLIB) ........................................................ 713
mlib_ImageZoomTranslateToGray(3MLIB) .......................................................... 714
mlib_malloc(3MLIB) ............................................................................................ 724
mlib_MatrixAddS_U8_Mod(3MLIB) ...................................................................... 725
mlib_MatrixAddS_U8_U8_Mod(3MLIB) ................................................................. 726
mlib_MatrixAdd_U8_Mod(3MLIB) ........................................................................ 727
mlib_MatrixAdd_U8_U8_Mod(3MLIB) ................................................................. 728
mlib_MatrixAve_U8(3MLIB) ............................................................................... 729
mlib_MatrixAve_U8_U8(3MLIB) ........................................................................... 730
Multimedia Library Functions - Part 4

mlib_MatrixMaximumMag_U8C(3MLIB) ............................................................... 740
mlib_MatrixMaximum_U8(3MLIB) ................................................................... 741
mlib_MatrixMinimumMag_U8C(3MLIB) .......................................................... 743
mlib_MatrixMinimum_U8(3MLIB) .................................................................. 744
mlib_MatrixMulShift_S16_S16_Mod(3MLIB) ...................................................... 746
mlib_MatrixMulSShift_U8_Mod(3MLIB) .......................................................... 750
mlib_MatrixMulSShift_U8_U8_Mod(3MLIB) ....................................................... 753
mlib_MatrixMulS_U8_Mod(3MLIB) ................................................................. 756
mlib_MatrixMulS_U8_U8_Mod(3MLIB) ............................................................ 758
mlib_MatrixMul_U8_U8_Mod(3MLIB) .............................................................. 761
mlib_MatrixScale_U8_Mod(3MLIB) ................................................................. 765
mlib_MatrixScale_U8_U8_Mod(3MLIB) ............................................................ 767
mlib_MatrixSubS_U8_Mod(3MLIB) ................................................................. 771
mlib_MatrixSubS_U8_U8_Mod(3MLIB) ............................................................ 773
mlib_MatrixSub_U8_Mod(3MLIB) ................................................................. 776
mlib_MatrixSub_U8_U8_Mod(3MLIB) ............................................................ 778
mlib_MatrixTranspose_U8(3MLIB) ............................................................... 781
mlib_MatrixTranspose_U8_U8(3MLIB) ........................................................... 783
mlib_MatrixUnit_U8(3MLIB) ....................................................................... 785
mlib_memcpy(3MLIB) .................................................................................. 787
mlib_memmove(3MLIB) ................................................................................ 788
mlib_memset(3MLIB) .................................................................................. 789
mlib_realloc(3MLIB) .................................................................................. 790
mlib_SignalADPCM2Bits2Linear(3MLIB) ...................................................... 791
mlib_SignalADPCM3Bits2Linear(3MLIB) ...................................................... 792
mlib_SignalADPCM4Bits2Linear(3MLIB) ...................................................... 793
mlib_SignalADPCM5Bits2Linear(3MLIB) ...................................................... 794
mlib_SignalADPCMFree(3MLIB) ................................................................ 795
mlib_SignalADPCMInit(3MLIB) ................................................................. 796
mlib_SignalALaw2Linear(3MLIB) ................................................................. 797
mlib_SignalALaw2uLaw(3MLIB) .................................................................. 798
mlib_SignalAutoCorrel_S16(3MLIB) ........................................................... 799
mlib_SignalCepstral_F32(3MLIB) ............................................................... 801
### Multimedia Library Functions - Part 5

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlibSignalIFFT_2 (3MLIB)</td>
<td>948</td>
</tr>
<tr>
<td>mlibSignalIFFT_3 (3MLIB)</td>
<td>951</td>
</tr>
<tr>
<td>mlibSignalIFFT_4 (3MLIB)</td>
<td>954</td>
</tr>
<tr>
<td>mlibSignalIFFTW_1 (3MLIB)</td>
<td>957</td>
</tr>
<tr>
<td>mlibSignalIFFTW_2 (3MLIB)</td>
<td>961</td>
</tr>
<tr>
<td>mlibSignalIFFTW_3 (3MLIB)</td>
<td>962</td>
</tr>
<tr>
<td>mlibSignalIFFTW_4 (3MLIB)</td>
<td>965</td>
</tr>
<tr>
<td>mlibSignalIIR_Bquad_S16_S16_Sat (3MLIB)</td>
<td>968</td>
</tr>
<tr>
<td>mlibSignalIIRFree_Bquad_F32_F32 (3MLIB)</td>
<td>971</td>
</tr>
<tr>
<td>mlibSignalIIRFree_Bquad_F32_S16 (3MLIB)</td>
<td>973</td>
</tr>
<tr>
<td>mlibSignalIIRFree_Bquad_S16_S16 (3MLIB)</td>
<td>974</td>
</tr>
<tr>
<td>mlibSignalIIRFree_P4_F32_F32 (3MLIB)</td>
<td>975</td>
</tr>
<tr>
<td>mlibSignalIIRFree_P4_S16_S16 (3MLIB)</td>
<td>976</td>
</tr>
<tr>
<td>mlibSignalIIRInit_Bquad_F32_F32 (3MLIB)</td>
<td>977</td>
</tr>
<tr>
<td>mlibSignalIIRInit_Bquad_S16_S16 (3MLIB)</td>
<td>978</td>
</tr>
<tr>
<td>mlibSignalIIRInit_P4_F32_F32 (3MLIB)</td>
<td>979</td>
</tr>
<tr>
<td>mlibSignalIIRInit_P4_S16_S16 (3MLIB)</td>
<td>980</td>
</tr>
<tr>
<td>mlibSignalIIR_P4_S16_S16_Sat (3MLIB)</td>
<td>981</td>
</tr>
<tr>
<td>mlibSignalIMDCT_D64 (3MLIB)</td>
<td>984</td>
</tr>
<tr>
<td>mlibSignalIMDCT_F32 (3MLIB)</td>
<td>985</td>
</tr>
<tr>
<td>mlibSignalIMDCTSplit_D64 (3MLIB)</td>
<td>986</td>
</tr>
<tr>
<td>mlibSignalIMDCTSplit_F32 (3MLIB)</td>
<td>987</td>
</tr>
<tr>
<td>mlibSignalLimit (3MLIB)</td>
<td>988</td>
</tr>
<tr>
<td>mlibSignalLinear2ADPCM2Bits (3MLIB)</td>
<td>990</td>
</tr>
<tr>
<td>mlibSignalLinear2ADPCM3Bits (3MLIB)</td>
<td>991</td>
</tr>
<tr>
<td>mlibSignalLinear2ADPCM4Bits (3MLIB)</td>
<td>992</td>
</tr>
<tr>
<td>mlibSignalLinear2ADPCM5Bits (3MLIB)</td>
<td>993</td>
</tr>
<tr>
<td>mlibSignalLinear2ALaw (3MLIB)</td>
<td>994</td>
</tr>
<tr>
<td>mlibSignalLinear2uLaw (3MLIB)</td>
<td>995</td>
</tr>
<tr>
<td>mlibSignalLMSFilter (3MLIB)</td>
<td>996</td>
</tr>
<tr>
<td>mlibSignalLPC2Cepstral_F32 (3MLIB)</td>
<td>999</td>
</tr>
<tr>
<td>mlibSignalLPC2Cepstral_S16 (3MLIB)</td>
<td>1001</td>
</tr>
<tr>
<td>mlibSignalLPC2Cepstral_S16_Adp (3MLIB)</td>
<td>1003</td>
</tr>
<tr>
<td>mlibSignalLPC2LSP_F32 (3MLIB)</td>
<td>1005</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>mlib_SignalLPC2LSP_S16(3MLIB)</td>
<td>1007</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrel_F32(3MLIB)</td>
<td>1009</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrelFree_S16(3MLIB)</td>
<td>1011</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrelGetEnergy_F32(3MLIB)</td>
<td>1012</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrelGetEnergy_S16(3MLIB)</td>
<td>1014</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrelGetPARCOR_F32(3MLIB)</td>
<td>1016</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrelGetPARCOR_S16(3MLIB)</td>
<td>1018</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrelInit_S16(3MLIB)</td>
<td>1020</td>
</tr>
<tr>
<td>mlib_SignalLPCAutoCorrel_S16(3MLIB)</td>
<td>1021</td>
</tr>
<tr>
<td>mlib_SignalLPCCovariance_F32(3MLIB)</td>
<td>1023</td>
</tr>
<tr>
<td>mlib_SignalLPCCovarianceFree_S16(3MLIB)</td>
<td>1025</td>
</tr>
<tr>
<td>mlib_SignalLPCCovarianceInit_S16(3MLIB)</td>
<td>1026</td>
</tr>
<tr>
<td>mlib_SignalLPCCovariance_S16(3MLIB)</td>
<td>1027</td>
</tr>
<tr>
<td>mlib_SignalLPCEncoded_F32(3MLIB)</td>
<td>1029</td>
</tr>
<tr>
<td>mlib_SignalLPCEncodedFree_S16(3MLIB)</td>
<td>1031</td>
</tr>
<tr>
<td>mlib_SignalLPCEncodedInit_S16(3MLIB)</td>
<td>1032</td>
</tr>
<tr>
<td>mlib_SignalLPCEncoded_S16(3MLIB)</td>
<td>1033</td>
</tr>
<tr>
<td>mlib_SignalLPCEncoded_S16_Adp(3MLIB)</td>
<td>1035</td>
</tr>
<tr>
<td>mlib_SignalLPCPerceptWeight_F32(3MLIB)</td>
<td>1037</td>
</tr>
<tr>
<td>mlib_SignalLPCPerceptWeightFree_S16(3MLIB)</td>
<td>1039</td>
</tr>
<tr>
<td>mlib_SignalLPCPerceptWeightInit_S16(3MLIB)</td>
<td>1041</td>
</tr>
<tr>
<td>mlib_SignalLPCPerceptWeight_S16(3MLIB)</td>
<td>1043</td>
</tr>
<tr>
<td>mlib_SignalLPCPitchAnalyze_F32(3MLIB)</td>
<td>1045</td>
</tr>
<tr>
<td>mlib_SignalLPCPitchAnalyze_s16(3MLIB)</td>
<td>1047</td>
</tr>
<tr>
<td>mlib_SignalLSP2LPC_F32(3MLIB)</td>
<td>1049</td>
</tr>
<tr>
<td>mlib_SignalLSP2LPC_S16(3MLIB)</td>
<td>1051</td>
</tr>
<tr>
<td>mlib_SignalMelCepstral_F32(3MLIB)</td>
<td>1053</td>
</tr>
<tr>
<td>mlib_SignalMelCepstralFree_S16(3MLIB)</td>
<td>1055</td>
</tr>
<tr>
<td>mlib_SignalMelCepstralInit_S16(3MLIB)</td>
<td>1057</td>
</tr>
<tr>
<td>mlib_SignalMelCepstral_S16(3MLIB)</td>
<td>1059</td>
</tr>
<tr>
<td>mlib_SignalMelCepstral_S16_Adp(3MLIB)</td>
<td>1061</td>
</tr>
<tr>
<td>mlib_SignalMerge_F32S_F32(3MLIB)</td>
<td>1063</td>
</tr>
<tr>
<td>mlib_SignalMerge_S16S_S16(3MLIB)</td>
<td>1065</td>
</tr>
<tr>
<td>mlib_SignalMulBartlett_F32(3MLIB)</td>
<td>1067</td>
</tr>
<tr>
<td>mlib_SignalMulBartlett_F32_F32(3MLIB)</td>
<td>1069</td>
</tr>
<tr>
<td>mlib_SignalMulBartlett_S16(3MLIB)</td>
<td>1071</td>
</tr>
<tr>
<td>mlib_SignalMulBartlett_S16_S16(3MLIB)</td>
<td>1073</td>
</tr>
<tr>
<td>mlib_SignalMulBlackman_F32(3MLIB)</td>
<td>1075</td>
</tr>
<tr>
<td>mlib_SignalMulBlackman_F32_F32(3MLIB)</td>
<td>1077</td>
</tr>
<tr>
<td>mlib_SignalMulBlackman_S16(3MLIB)</td>
<td>1079</td>
</tr>
<tr>
<td>mlib_SignalMulBlackman_S16_S16(3MLIB)</td>
<td>1081</td>
</tr>
<tr>
<td>Function Name</td>
<td>Line Number</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mlib_SignalMul_F32(3MLIB)</td>
<td>1062</td>
</tr>
<tr>
<td>mlib_SignalMul_F32_F32(3MLIB)</td>
<td>1063</td>
</tr>
<tr>
<td>mlib_SignalMulHamming_F32(3MLIB)</td>
<td>1064</td>
</tr>
<tr>
<td>mlib_SignalMulHamming_F32_F32(3MLIB)</td>
<td>1065</td>
</tr>
<tr>
<td>mlib_SignalMulHamming_S16(3MLIB)</td>
<td>1066</td>
</tr>
<tr>
<td>mlib_SignalMulHamming_S16_S16(3MLIB)</td>
<td>1067</td>
</tr>
<tr>
<td>mlib_SignalMulHanning_F32(3MLIB)</td>
<td>1068</td>
</tr>
<tr>
<td>mlib_SignalMulHanning_F32_F32(3MLIB)</td>
<td>1069</td>
</tr>
<tr>
<td>mlib_SignalMulHanning_S16(3MLIB)</td>
<td>1070</td>
</tr>
<tr>
<td>mlib_SignalMulHanning_S16_S16(3MLIB)</td>
<td>1071</td>
</tr>
<tr>
<td>mlib_SignalMulKaiser_F32(3MLIB)</td>
<td>1072</td>
</tr>
<tr>
<td>mlib_SignalMulKaiser_F32_F32(3MLIB)</td>
<td>1073</td>
</tr>
<tr>
<td>mlib_SignalMulKaiser_S16(3MLIB)</td>
<td>1074</td>
</tr>
<tr>
<td>mlib_SignalMulKaiser_S16_S16(3MLIB)</td>
<td>1075</td>
</tr>
<tr>
<td>mlib_SignalMulRectangular_F32(3MLIB)</td>
<td>1076</td>
</tr>
<tr>
<td>mlib_SignalMulRectangular_F32_F32(3MLIB)</td>
<td>1077</td>
</tr>
<tr>
<td>mlib_SignalMulRectangular_S16(3MLIB)</td>
<td>1078</td>
</tr>
<tr>
<td>mlib_SignalMulRectangular_S16_S16(3MLIB)</td>
<td>1079</td>
</tr>
<tr>
<td>mlib_SignalMul_S16_S16_Sat(3MLIB)</td>
<td>1080</td>
</tr>
<tr>
<td>mlib_SignalMul_S16_Sat(3MLIB)</td>
<td>1081</td>
</tr>
<tr>
<td>mlib_SignalMulSAdd_F32(3MLIB)</td>
<td>1082</td>
</tr>
<tr>
<td>mlib_SignalMulSAdd_F32_F32(3MLIB)</td>
<td>1083</td>
</tr>
<tr>
<td>mlib_SignalMulSAdd_S16_S16_Sat(3MLIB)</td>
<td>1084</td>
</tr>
<tr>
<td>mlib_SignalMulSAdd_S16_Sat(3MLIB)</td>
<td>1085</td>
</tr>
<tr>
<td>mlib_SignalMulS_F32(3MLIB)</td>
<td>1086</td>
</tr>
<tr>
<td>mlib_SignalMulS_F32_F32(3MLIB)</td>
<td>1087</td>
</tr>
<tr>
<td>mlib_SignalMulShift_S16_S16_Sat(3MLIB)</td>
<td>1088</td>
</tr>
<tr>
<td>mlib_SignalMulShift_S16_Sat(3MLIB)</td>
<td>1089</td>
</tr>
<tr>
<td>mlib_SignalMulS_S16_S16_Sat(3MLIB)</td>
<td>1090</td>
</tr>
<tr>
<td>mlib_SignalMulSShiftAdd_S16_S16_Sat(3MLIB)</td>
<td>1091</td>
</tr>
<tr>
<td>mlib_SignalMulSShiftAdd_S16_Sat(3MLIB)</td>
<td>1092</td>
</tr>
<tr>
<td>mlib_SignalMulSShiftAdd_S16_S16_Sat(3MLIB)</td>
<td>1093</td>
</tr>
<tr>
<td>mlib_SignalMulSShiftAdd_S16_Sat(3MLIB)</td>
<td>1094</td>
</tr>
<tr>
<td>mlib_SignalMulSShift_S16_Sat(3MLIB)</td>
<td>1095</td>
</tr>
<tr>
<td>mlib_SignalMulWindow_F32(3MLIB)</td>
<td>1096</td>
</tr>
<tr>
<td>mlib_SignalMulWindow_F32_F32(3MLIB)</td>
<td>1097</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>mlib_SignalMulWindow_F32S(3MLIB)</td>
<td>1098</td>
</tr>
<tr>
<td>mlib_SignalMulWindow_F32S_F32S(3MLIB)</td>
<td>1099</td>
</tr>
<tr>
<td>mlib_SignalMulWindow_S16(3MLIB)</td>
<td>1100</td>
</tr>
<tr>
<td>mlib_SignalMulWindow_S16_S16(3MLIB)</td>
<td>1101</td>
</tr>
<tr>
<td>mlib_SignalNLMSFilter(3MLIB)</td>
<td>1102</td>
</tr>
<tr>
<td>mlib_SignalQuant2_S16_F32(3MLIB)</td>
<td>1105</td>
</tr>
<tr>
<td>mlib_SignalQuant2_S16S_F32S(3MLIB)</td>
<td>1106</td>
</tr>
<tr>
<td>mlib_SignalQuant_S16_F32(3MLIB)</td>
<td>1107</td>
</tr>
<tr>
<td>mlib_SignalQuant_S16S_F32S(3MLIB)</td>
<td>1108</td>
</tr>
<tr>
<td>mlib_SignalQuant_U8_F32(3MLIB)</td>
<td>1109</td>
</tr>
<tr>
<td>mlib_SignalQuant_U8_S16(3MLIB)</td>
<td>1110</td>
</tr>
<tr>
<td>mlib_SignalQuant_U8S_F32S(3MLIB)</td>
<td>1111</td>
</tr>
<tr>
<td>mlib_SignalReSampleFIR_F32_F32(3MLIB)</td>
<td>1112</td>
</tr>
<tr>
<td>mlib_SignalReSampleFIR_F32S_F32S(3MLIB)</td>
<td>1113</td>
</tr>
<tr>
<td>mlib_SignalReSampleFIRFree_F32_F32(3MLIB)</td>
<td>1114</td>
</tr>
<tr>
<td>mlib_SignalReSampleFIRFree_F32S_F32S(3MLIB)</td>
<td>1115</td>
</tr>
<tr>
<td>mlib_SignalReSampleFIRInit_S16_S16(3MLIB)</td>
<td>1116</td>
</tr>
<tr>
<td>mlib_SignalReSampleFIR_S16_S16_Sat(3MLIB)</td>
<td>1117</td>
</tr>
<tr>
<td>mlib_SignalSineWave_F32(3MLIB)</td>
<td>1118</td>
</tr>
<tr>
<td>mlib_SignalSineWaveFree_F32(3MLIB)</td>
<td>1119</td>
</tr>
<tr>
<td>mlib_SignalSineWaveFree_S16(3MLIB)</td>
<td>1120</td>
</tr>
<tr>
<td>mlib_SignalSineWaveInit_F32(3MLIB)</td>
<td>1121</td>
</tr>
<tr>
<td>mlib_SignalSineWaveInit_S16(3MLIB)</td>
<td>1122</td>
</tr>
<tr>
<td>mlib_SignalSineWave_S16(3MLIB)</td>
<td>1123</td>
</tr>
<tr>
<td>mlib_SignalSplit_F32_F32S(3MLIB)</td>
<td>1124</td>
</tr>
<tr>
<td>mlib_SignalSplit_S16_S16S(3MLIB)</td>
<td>1125</td>
</tr>
<tr>
<td>mlib_SignaluLaw2ALaw(3MLIB)</td>
<td>1126</td>
</tr>
<tr>
<td>mlib_SignaluLaw2Linear(3MLIB)</td>
<td>1127</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIR_F32_F32(3MLIB)</td>
<td>1128</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIR_F32S_F32S(3MLIB)</td>
<td>1129</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRFree_F32_F32(3MLIB)</td>
<td>1130</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRFree_F32S_F32S(3MLIB)</td>
<td>1131</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRInit_S16_S16(3MLIB)</td>
<td>1132</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRInit_F32_F32(3MLIB)</td>
<td>1133</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRInit_F32S_F32S(3MLIB)</td>
<td>1134</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRInit_F32S_F32S(3MLIB)</td>
<td>1135</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRInit_F32S_F32S(3MLIB)</td>
<td>1136</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIRInit_S16_S16(3MLIB)</td>
<td>1137</td>
</tr>
<tr>
<td>mlib_SignalUpSampleFIR_S16_S16_Sat(3MLIB)</td>
<td>1138</td>
</tr>
<tr>
<td>mlib_SignalUpSample_S16_S16(3MLIB)</td>
<td>1139</td>
</tr>
<tr>
<td>mlib_SignalWhiteNoise_F32(3MLIB)</td>
<td>1141</td>
</tr>
<tr>
<td>mlib_SignalWhiteNoiseFree_F32(3MLIB)</td>
<td>1142</td>
</tr>
<tr>
<td>mlib_SignalWhiteNoiseFree_S16(3MLIB)</td>
<td>1143</td>
</tr>
<tr>
<td>mlib_SignalWhiteNoiseInit_F32(3MLIB)</td>
<td>1144</td>
</tr>
<tr>
<td>mlib_SignalWhiteNoiseInit_S16(3MLIB)</td>
<td>1145</td>
</tr>
<tr>
<td>mlib_SignalWhiteNoise_S16(3MLIB)</td>
<td>1146</td>
</tr>
<tr>
<td>mlib_VectorAddS_U8_Mod(3MLIB)</td>
<td>1147</td>
</tr>
<tr>
<td>mlib_VectorAddS_U8_U8_Mod(3MLIB)</td>
<td>1149</td>
</tr>
<tr>
<td>mlib_VectorAdd_U8_Mod(3MLIB)</td>
<td>1152</td>
</tr>
<tr>
<td>mlib_VectorAdd_U8_U8_Mod(3MLIB)</td>
<td>1154</td>
</tr>
<tr>
<td>mlib_VectorAng_U8C(3MLIB)</td>
<td>1157</td>
</tr>
<tr>
<td>mlib_VectorAve_U8(3MLIB)</td>
<td>1158</td>
</tr>
<tr>
<td>mlib_VectorAve_U8_U8(3MLIB)</td>
<td>1160</td>
</tr>
<tr>
<td>mlib_VectorConjRev_S8C_S8C_Sat(3MLIB)</td>
<td>1162</td>
</tr>
<tr>
<td>mlib_VectorConj_S8C_S8C_Sat(3MLIB)</td>
<td>1163</td>
</tr>
<tr>
<td>mlib_VectorConj_S8C_Sat(3MLIB)</td>
<td>1164</td>
</tr>
<tr>
<td>mlib_VectorConjSymExt_S8C_S8C_Sat(3MLIB)</td>
<td>1165</td>
</tr>
<tr>
<td>mlib_VectorConvert_U8_S8_Mod(3MLIB)</td>
<td>1168</td>
</tr>
<tr>
<td>mlib_VectorCopy_U8(3MLIB)</td>
<td>1173</td>
</tr>
<tr>
<td>mlib_VectorDistance_U8_Sat(3MLIB)</td>
<td>1175</td>
</tr>
<tr>
<td>mlib_VectorDotProd_U8_Sat(3MLIB)</td>
<td>1176</td>
</tr>
<tr>
<td>mlib_VectorMag_U8C(3MLIB)</td>
<td>1178</td>
</tr>
<tr>
<td>mlib_VectorMaximumMag_U8C(3MLIB)</td>
<td>1179</td>
</tr>
<tr>
<td>mlib_VectorMaximum_U8(3MLIB)</td>
<td>1180</td>
</tr>
<tr>
<td>mlib_VectorMerge_U8C_U8(3MLIB)</td>
<td>1181</td>
</tr>
<tr>
<td>mlib_VectorMinimumMag_U8C(3MLIB)</td>
<td>1183</td>
</tr>
<tr>
<td>mlib_VectorMinimum_U8(3MLIB)</td>
<td>1184</td>
</tr>
<tr>
<td>mlib_VectorMulMShift_S16_S16_Mod(3MLIB)</td>
<td>1185</td>
</tr>
<tr>
<td>mlib_VectorMulM_U8_U8_Mod(3MLIB)</td>
<td>1187</td>
</tr>
<tr>
<td>mlib_VectorMulSAdd_U8_Mod(3MLIB)</td>
<td>1190</td>
</tr>
</tbody>
</table>

**Multimedia Library Functions - Part 6**

*mlib_VectorConvert_U8_S8_Mod(3MLIB)* .................................................. 1168
*mlib_VectorCopy_U8(3MLIB) ................................................................. 1173
*mlib_VectorDistance_U8_Sat(3MLIB) .................................................... 1175
*mlib_VectorDotProd_U8_Sat(3MLIB) ....................................................... 1176
*mlib_VectorMag_U8C(3MLIB) ................................................................. 1178
*mlib_VectorMaximumMag_U8C(3MLIB) .................................................... 1179
*mlib_VectorMaximum_U8(3MLIB) ........................................................... 1180
*mlib_VectorMerge_U8C_U8(3MLIB) ........................................................ 1181
*mlib_VectorMinimumMag_U8C(3MLIB) .................................................... 1183
*mlib_VectorMinimum_U8(3MLIB) ........................................................... 1184
*mlib_VectorMulMShift_S16_S16_Mod(3MLIB) ........................................... 1185
*mlib_VectorMulM_U8_U8_Mod(3MLIB) ....................................................... 1187
*mlib_VectorMulSAdd_U8_Mod(3MLIB) ..................................................... 1190
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlib_VideoColorARGB2JFIFYCC422(3MLIB)</td>
<td>AMGB to YCC422</td>
</tr>
<tr>
<td>mlib_VideoColorARGB2JFIFYCC444(3MLIB)</td>
<td>AMGB to YCC444</td>
</tr>
<tr>
<td>mlib_VideoColorARGB2RGB(3MLIB)</td>
<td>AMGB to RGB</td>
</tr>
<tr>
<td>mlib_VideoColorBGR2JFIFYCC420(3MLIB)</td>
<td>BGRA to YCC420</td>
</tr>
<tr>
<td>mlib_VideoColorBGR2JFIFYCC422(3MLIB)</td>
<td>BGRA to YCC422</td>
</tr>
<tr>
<td>mlib_VideoColorBGR2JFIFYCC444(3MLIB)</td>
<td>BGRA to YCC444</td>
</tr>
<tr>
<td>mlib_VideoColorBGR2JFIFYCC444_S16(3MLIB)</td>
<td>BGRA to YCC444, S16</td>
</tr>
<tr>
<td>mlib_VideoColorBGRAint_to_ABGRint(3MLIB)</td>
<td>BGRA to ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorBGRint_to_ABGRint(3MLIB)</td>
<td>BGR to ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorBlendABGR(3MLIB)</td>
<td>Blend ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorBlendABGR_Inp(3MLIB)</td>
<td>ABGR input for blend ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorCMYK2JFIFYCC444(3MLIB)</td>
<td>CMYK to YCC444</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2ABGR420(3MLIB)</td>
<td>JFIF to ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2ABGR422(3MLIB)</td>
<td>JFIF to ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2RGB420(3MLIB)</td>
<td>JFIF to RGB</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2RGB420_Nearest(3MLIB)</td>
<td>JFIF to RGB, Nearest</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2RGB422(3MLIB)</td>
<td>JFIF to RGB</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2RGB422_Nearest(3MLIB)</td>
<td>JFIF to RGB, Nearest</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2RGB444(3MLIB)</td>
<td>JFIF to RGB</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2RGB444_S16(3MLIB)</td>
<td>JFIF to RGB, S16</td>
</tr>
<tr>
<td>mlib_VideoColorJFIFYCC2CMYK444(3MLIB)</td>
<td>JFIF to CMYK</td>
</tr>
<tr>
<td>mlib_VideoColorMerge2(3MLIB)</td>
<td>Merge 2</td>
</tr>
<tr>
<td>mlib_VideoColorMerge2_S16(3MLIB)</td>
<td>Merge 2, S16</td>
</tr>
<tr>
<td>mlib_VideoColorMerge3(3MLIB)</td>
<td>Merge 3</td>
</tr>
<tr>
<td>mlib_VideoColorMerge3_S16(3MLIB)</td>
<td>Merge 3, S16</td>
</tr>
<tr>
<td>mlib_VideoColorMerge4(3MLIB)</td>
<td>Merge 4</td>
</tr>
<tr>
<td>mlib_VideoColorMerge4_S16(3MLIB)</td>
<td>Merge 4, S16</td>
</tr>
<tr>
<td>mlib_VideoColorResizeABGR(3MLIB)</td>
<td>Resize ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorRGB2ABGR(3MLIB)</td>
<td>RGB to ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorRGB2ARGB(3MLIB)</td>
<td>RGB to ARGb</td>
</tr>
<tr>
<td>mlib_VideoColorRGB2JFIFYCC420(3MLIB)</td>
<td>RGB to JFIFYCC420</td>
</tr>
<tr>
<td>mlib_VideoColorRGB2JFIFYCC422(3MLIB)</td>
<td>RGB to JFIFYCC422</td>
</tr>
<tr>
<td>mlib_VideoColorRGB2JFIFYCC444(3MLIB)</td>
<td>RGB to JFIFYCC444</td>
</tr>
<tr>
<td>mlib_VideoColorRGB2JFIFYCC444_S16(3MLIB)</td>
<td>RGB to JFIFYCC444, S16</td>
</tr>
<tr>
<td>mlib_VideoColorRGBAint_to_ABGRint(3MLIB)</td>
<td>RGBA to ABGR</td>
</tr>
<tr>
<td>mlib_VideoColorRGBint_to_ABGRint(3MLIB)</td>
<td>BGR to ABGR</td>
</tr>
</tbody>
</table>

25
mlib_VideoColorRGBint_to_BGRArInt(3MLIB) ................................................................. 1295
mlib_VideoColorRGBseq_to_ABGrInt(3MLIB) ................................................................ 1297
mlib_VideoColorRGBXint_to_ABGrInt(3MLIB) ................................................................ 1299
mlib_VideoColorRGBXint_to_ARGrInt(3MLIB) ................................................................. 1301
mlib_VideoColorSplit2(3MLIB) .................................................................................. 1302
mlib_VideoColorSplit2_S16(3MLIB) .............................................................................. 1303
mlib_VideoColorSplit3(3MLIB) .................................................................................. 1304
mlib_VideoColorSplit3_S16(3MLIB) .............................................................................. 1305
mlib_VideoColorSplit4(3MLIB) .................................................................................. 1306
mlib_VideoColorSplit4_S16(3MLIB) .............................................................................. 1307
mlib_VideoColorUYV444int_to_ABGrInt(3MLIB) ............................................................. 1308
mlib_VideoColorUYV444int_to_ARGrInt(3MLIB) ............................................................. 1310
mlib_VideoColorUYV444int_to_UVYV422int(3MLIB) .................................................... 1312
mlib_VideoColorUYV444int_to_YVYV422int(3MLIB) .................................................... 1314
mlib_VideoColorUYVY422int_to_ABGrInt(3MLIB) .......................................................... 1316
mlib_VideoColorUYVY422int_to_ARGrInt(3MLIB) .......................................................... 1318
mlib_VideoColorXRGBint_to_ABGrInt(3MLIB) .............................................................. 1320
mlib_VideoColorXRGBint_to_ARGrInt(3MLIB) .............................................................. 1321
mlib_VideoColorYUV2ABGR411(3MLIB) ....................................................................... 1322
mlib_VideoColorYUV2ABGR420(3MLIB) ....................................................................... 1324
mlib_VideoColorYUV2ABGR420_W(3MLIB) .................................................................. 1326
mlib_VideoColorYUV2ABGR420_Wx2(3MLIB) ............................................................... 1328
mlib_VideoColorYUV2ABGR420_Wx3(3MLIB) ............................................................... 1330
mlib_VideoColorYUV2ABGR420_Wx4(3MLIB) ............................................................... 1332
mlib_VideoColorYUV2ABGR420_Wx3(3MLIB) ............................................................... 1334
mlib_VideoColorYUV2ABGR422(3MLIB) ...................................................................... 1336
mlib_VideoColorYUV2ABGR444(3MLIB) ...................................................................... 1338
mlib_VideoColorYUV2ARGB411(3MLIB) .......................................................... 1339
mlib_VideoColorYUV2ARGB420(3MLIB) ................................................................ 1341
mlib_VideoColorYUV2ARGB422(3MLIB) ................................................................ 1343
mlib_VideoColorYUV2ARGB444(3MLIB) ................................................................ 1345
mlib_VideoColorYUV2RGB411(3MLIB) ................................................................ 1346
mlib_VideoColorYUV2RGB420(3MLIB) ................................................................ 1348
mlib_VideoColorYUV2RGB422(3MLIB) ................................................................ 1350
mlib_VideoColorYUV2RGB444(3MLIB) ................................................................ 1352
mlib_VideoColorYUV411seq_to_ABGrInt(3MLIB) ........................................................... 1353
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Notes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlib_VideoDCT8x8_S16_S16_B10 NA(3MLIB)</td>
<td></td>
<td>1418</td>
</tr>
<tr>
<td>mlib_VideoDCT8x8_S16_S16_B12(3MLIB)</td>
<td></td>
<td>1419</td>
</tr>
<tr>
<td>mlib_VideoDCT8x8_S16_U8(3MLIB)</td>
<td></td>
<td>1420</td>
</tr>
<tr>
<td>mlib_VideoDCT8x8_S16_U8 NA(3MLIB)</td>
<td></td>
<td>1421</td>
</tr>
<tr>
<td>mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12(3MLIB)</td>
<td></td>
<td>1422</td>
</tr>
<tr>
<td>mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12 NA(3MLIB)</td>
<td></td>
<td>1423</td>
</tr>
<tr>
<td>mlib_VideoDeQuantizeIDCT8x8_U8_S16(3MLIB)</td>
<td></td>
<td>1424</td>
</tr>
<tr>
<td>mlib_VideoDeQuantizeIDCT8x8_U8_S16 NA(3MLIB)</td>
<td></td>
<td>1425</td>
</tr>
<tr>
<td>mlib_VideoDeQuantizeInit_S16(3MLIB)</td>
<td></td>
<td>1426</td>
</tr>
<tr>
<td>mlib_VideoDeQuantize_S16(3MLIB)</td>
<td></td>
<td>1427</td>
</tr>
<tr>
<td>mlib_VideoDownSample420(3MLIB)</td>
<td></td>
<td>1428</td>
</tr>
<tr>
<td>mlib_VideoDownSample420_S16(3MLIB)</td>
<td></td>
<td>1429</td>
</tr>
<tr>
<td>mlib_VideoDownSample422(3MLIB)</td>
<td></td>
<td>1430</td>
</tr>
<tr>
<td>mlib_VideoDownSample422_S16(3MLIB)</td>
<td></td>
<td>1431</td>
</tr>
<tr>
<td>mlib_VideoH263OverlappedMC_S16_U8(3MLIB)</td>
<td></td>
<td>1432</td>
</tr>
<tr>
<td>mlib_VideoH263OverlappedMC_U8_U8(3MLIB)</td>
<td></td>
<td>1433</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_S16_S16_B12(3MLIB)</td>
<td></td>
<td>1438</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_S16_S16_B12 NA(3MLIB)</td>
<td></td>
<td>1440</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_S16_S16_DC(3MLIB)</td>
<td></td>
<td>1442</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB)</td>
<td></td>
<td>1443</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_S16_S16_Q1 Mismatch(3MLIB)</td>
<td></td>
<td>1444</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_U8_S16(3MLIB)</td>
<td></td>
<td>1445</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_U8_S16_DC(3MLIB)</td>
<td></td>
<td>1446</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_U8_S16 NA(3MLIB)</td>
<td></td>
<td>1447</td>
</tr>
<tr>
<td>mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB)</td>
<td></td>
<td>1448</td>
</tr>
<tr>
<td>mlib_VideoIDCT_IEEE_S16_S16(3MLIB)</td>
<td></td>
<td>1449</td>
</tr>
<tr>
<td>mlib_VideoInterpAveX_U8_U8_16x16(3MLIB)</td>
<td></td>
<td>1450</td>
</tr>
<tr>
<td>mlib_VideoInterpAveX_U8_U8(3MLIB)</td>
<td></td>
<td>1452</td>
</tr>
<tr>
<td>mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB)</td>
<td></td>
<td>1454</td>
</tr>
<tr>
<td>mlib_VideoInterpAveXY_U8_U8(3MLIB)</td>
<td></td>
<td>1456</td>
</tr>
<tr>
<td>mlib_VideoInterpAveY_U8_U8_16x16(3MLIB)</td>
<td></td>
<td>1458</td>
</tr>
<tr>
<td>mlib_VideoInterpAveY_U8_U8(3MLIB)</td>
<td></td>
<td>1460</td>
</tr>
<tr>
<td>mlib_VideoInterpX_S16_U8_16x16(3MLIB)</td>
<td></td>
<td>1462</td>
</tr>
<tr>
<td>mlib_VideoInterpX_S16_U8(3MLIB)</td>
<td></td>
<td>1464</td>
</tr>
<tr>
<td>mlib_VideoInterpX_U8_U8_16x16(3MLIB)</td>
<td></td>
<td>1466</td>
</tr>
<tr>
<td>mlib_VideoInterpX_U8_U8(3MLIB)</td>
<td></td>
<td>1468</td>
</tr>
</tbody>
</table>
mlib_VideoInterpXY_S16_U8_16x16(3MLIB) ........................................................... 1470
mlib_VideoInterpXY_S16_U8(3MLIB) ................................................................ 1472
mlib_VideoInterpXY_U8_U8_16x16(3MLIB) ......................................................... 1474
mlib_VideoInterpXY_U8_U8(3MLIB) ................................................................ 1476
mlib_VideoInterpX_Y_XY_U8_U8 ................................................................. 1478
mlib_VideoInterpY_S16_U8_16x16(3MLIB) ......................................................... 1479
mlib_VideoInterpY_S16_U8(3MLIB) ................................................................ 1481
mlib_VideoInterpY_U8_U8_16x16(3MLIB) ......................................................... 1483
mlib_VideoInterpY_U8_U8(3MLIB) ................................................................ 1485
mlib_VideoP64Decimate_U8_U8(3MLIB) ................................................................ 1487
mlib_VideoP64Loop_S16_U8(3MLIB) ................................................................ 1489
mlib_VideoP64Loop_U8_U8(3MLIB) ................................................................ 1491
mlib_VideoQuantizeInit_S16(3MLIB) ............................................................... 1493
mlib_VideoQuantize_S16(3MLIB) .................................................................... 1494
mlib_VideoReversibleColorRGB2YUV_U8_U8(3MLIB) ...................................... 1495
mlib_VideoReversibleColorYUV2RGB_U8_U8(3MLIB) ...................................... 1497
mlib_VideoSignMagnitudeConvert_S16(3MLIB) .............................................. 1499
mlib_VideoSignMagnitudeConvert_S32(3MLIB) ............................................. 1501
mlib_VideoSignMagnitudeConvert_S32_S32(3MLIB) ..................................... 1502
mlib_VideoSumAbsDiff(3MLIB) ..................................................................... 1503
mlib_VideoUpSample420(3MLIB) .................................................................... 1504
mlib_VideoUpSample420_Nearest(3MLIB) ....................................................... 1505
mlib_VideoUpSample420_Nearest_S16(3MLIB) ............................................. 1506
mlib_VideoUpSample420_S16(3MLIB) ............................................................... 1507
mlib_VideoUpSample422(3MLIB) .................................................................... 1508
mlib_VideoUpSample422_Nearest(3MLIB) ....................................................... 1509
mlib_VideoUpSample422_Nearest_S16(3MLIB) ............................................. 1510
mlib_VideoUpSample422_S16(3MLIB) ............................................................... 1511
mlib_VideoWaveletForwardTwoTenTrans(3MLIB) ........................................ 1512
mlib_VideoWaveletInverseTwoTenTrans(3MLIB) ......................................... 1513
mlib_VolumeFindMaxBMask_U8(3MLIB) ......................................................... 1514
mlib_VolumeFindMaxCMask_U8(3MLIB) ......................................................... 1515
mlib_VolumeFindMax_U8(3MLIB) .................................................................. 1516
mlib_VolumeRayCast_Blocked(3MLIB) .......................................................... 1517
mlib_VolumeRayCast_General(3MLIB) ............................................................ 1519
Preface

Both novice users and those familiar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9E describes the DDI (Device Driver Interface)/DKI (Driver/Kernel Interface), DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report,
there is no BUGS section. See the intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

NAME
This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS
This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

... Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, “filename...”.

| Separator. Only one of the arguments separated by this character can be specified at a time.

{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL
This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION
This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.

IOCTL
This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device).
ioctl calls are used for a particular class of devices all of which have an \texttt{io} ending, such as \texttt{mtio(7I)}.

**OPTIONS**

This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

**OPERANDS**

This section lists the command operands and describes how they affect the actions of the command.

**OUTPUT**

This section describes the output – standard output, standard error, or output files – generated by the command.

**RETURN VALUES**

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or \texttt{-1}, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared \texttt{void} do not return values, so they are not discussed in \texttt{RETURN VALUES}.

**ERRORS**

On failure, most functions place an error code in the global variable \texttt{errno} indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

**USAGE**

This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

**EXAMPLES**

This section provides examples of usage or of how to use a command or function. Wherever possible a complete
example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as example%, or if the user must be superuser, example#. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

ENVIRONMENT VARIABLES This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

FILES This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

ATTRIBUTES This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See attributes(5) for more information.

SEE ALSO This section lists references to other man pages, in-house documentation, and outside publications.

DIAGNOSTICS This section lists diagnostic messages with a brief explanation of the condition causing the error.

WARNINGS This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

NOTES This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS This section describes known bugs and, wherever possible, suggests workarounds.
REFERENCE

Multimedia Library Functions - Part 1
The `mlib_free()` function frees a block of bytes previously allocated by `mlib_malloc()` or `mlib_realloc()`.

This function is a wrapper of the standard C function `free()`.

### Parameters
The function takes the following arguments:
- **ptr**  Pointer to a previously allocated block.

### Return Values
None.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_malloc(3MLIB)`, `mlib_realloc(3MLIB)`, `malloc(3C)`, `attributes(5)`
Name  mlib_GraphicsBoundaryFill_8, mlib_GraphicsBoundaryFill_32 – boundary fill

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsBoundaryFill_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsBoundaryFill_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c, mlib_s32 c2);

Description  Each of these functions performs boundary fill.

Parameters  Each of the functions takes the following arguments:
  
  buffer  Pointer to the image into which the function is drawing.
  
  x  X coordinate of the starting point.
  
  y  Y coordinate of the starting point.
  
  c  Color used in the drawing.
  
  c2  Color that defines the filling boundary.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
mlib_GraphicsDrawArc, mlib_GraphicsDrawArc_8, mlib_GraphicsDrawArc_32,
mlib_GraphicsDrawArc_X_8, mlib_GraphicsDrawArc_X_32, mlib_GraphicsDrawArc_A_8,
mlib_GraphicsDrawArc_A_32, mlib_GraphicsDrawArc_B_8,
mlib_GraphicsDrawArc_B_32, mlib_GraphicsDrawArc_AB_8,
mlib_GraphicsDrawArc_AB_32 – draw arc

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawArc_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c);
mlib_status mlib_GraphicsDrawArc_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c);
mlib_status mlib_GraphicsDrawArc_X_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c,
mlib_s32 c2);
mlib_status mlib_GraphicsDrawArc_X_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c,
mlib_s32 c2);
mlib_status mlib_GraphicsDrawArc_A_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c);
mlib_status mlib_GraphicsDrawArc_A_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c);
mlib_status mlib_GraphicsDrawArc_B_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c,
mlib_s32 a);
mlib_status mlib_GraphicsDrawArc_B_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c,
mlib_s32 a);
mlib_status mlib_GraphicsDrawArc_AB_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c,
mlib_s32 a);
mlib_status mlib_GraphicsDrawArc_AB_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 r, mlib_f32 t1, mlib_f32 t2, mlib_s32 c,
mlib_s32 a);

Description Each of the mlib_GraphicsDrawArc_*( ) functions draws an arc with the center at \((x, y)\),
radius \(r\), start angle \(t1\), and end angle \(t2\).

Each of the mlib_GraphicsDrawArc_X_*( ) functions draws an arc in Xor mode as follows:
\[ \text{data}[x, y] \leftarrow c \oplus \text{c2} \]

Each of the mlib_GraphicsDrawArc_A_*( ) functions draws an arc with antialiasing.
Each of the \texttt{mlib\_GraphicsDrawArc\_B\_\*()} functions draws an arc with alpha blending as follows:
\[
data[x,y] = (data[x,y] * (255 - a) + c \times a) / 255
\]

Each of the \texttt{mlib\_GraphicsDrawArc\_AB\_\*()} functions draws an arc with antialiasing and alpha blending.

**Parameters**
Each of the functions takes some of the following arguments:
- \textit{buffer} Pointer to the image into which the function is drawing.
- \textit{x} X coordinate of the center.
- \textit{y} Y coordinate of the center.
- \textit{r} Radius of the arc.
- \textit{t1} Start angle of the arc in radians.
- \textit{t2} End angle of the arc in radians.
- \textit{c} Color used in the drawing.
- \textit{c2} Alternation color.
- \textit{a} Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values**
Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

**Attributes**
See attributes\texttt{(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**\texttt{mlib\_GraphicsDrawCircle\(3MLIB\)}, \texttt{mlib\_GraphicsDrawEllipse\(3MLIB\)}, attributes\texttt{(5)}

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawCircle_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsDrawCircle_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsDrawCircle_X_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawCircle_X_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawCircle_A_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsDrawCircle_A_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsDrawCircle_B_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawCircle_B_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawCircle_AB_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawCircle_AB_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);

Each of the mlib_GraphicsDrawCircle_*( ) functions draws a circle with the center at \((x, y)\) and radius \(r\).

Each of the mlib_GraphicsDrawCircle_X_*( ) functions draws a circle in Xor mode as follows:
\[
data[x, y] ^= c^c\]

Each of the mlib_GraphicsDrawCircle_A_*( ) functions draws a circle with antialiasing.

Each of the mlib_GraphicsDrawCircle_B_*( ) functions draws a circle with alpha blending as follows:
\[
data[x, y] = (data[x, y] * (255 - a) + c * a) / 255\]
Each of the `mlib_GraphicsDrawCircle_AB_*( )` functions draws a circle with antialiasing and alpha blending.

**Parameters** Each of the functions takes some of the following arguments:
- `buffer` Pointer to the image into which the function is drawing.
- `x` X coordinate of the center.
- `y` Y coordinate of the center.
- `r` Radius of the arc.
- `c` Color used in the drawing.
- `c2` Alternation color.
- `a` Alpha value for blending. $0 \leq a \leq 255$.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_GraphicsDrawArc(3MLIB), mlib_GraphicsDrawEllipse(3MLIB), attributes(5)`
Name  
mlib_GraphicsDrawEllipse_X_8, mlib_GraphicsDrawEllipse_X_32,
mlib_GraphicsDrawEllipse_A_8, mlib_GraphicsDrawEllipse_A_32,
mlib_GraphicsDrawEllipse_B_8, mlib_GraphicsDrawEllipse_B_32,

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawEllipse_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsDrawEllipse_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsDrawEllipse_X_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawEllipse_X_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawEllipse_A_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsDrawEllipse_A_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsDrawEllipse_B_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
mlib_status mlib_GraphicsDrawEllipse_B_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
mlib_status mlib_GraphicsDrawEllipse_AB_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
mlib_status mlib_GraphicsDrawEllipse_AB_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);

Description  
Each of the mlib_GraphicsDrawEllipse (*) functions draws an ellipse with the center at (x, y), major semiaxis a, and minor semiaxis b. The angle of the major semiaxis is t counterclockwise from the X axis.

Each of the mlib_GraphicsDrawEllipse_X_() functions draws an ellipse in Xor mode as follows:

\[ data[x,y] \^= c \^ c2 \]
Each of the `mlib_GraphicsDrawEllipse_A_*()` functions draws an ellipse with antialiasing.

Each of the `mlib_GraphicsDrawEllipse_B_*()` functions draws an ellipse with alpha blending as follows:

\[
data[x,y] = (data[x,y] \times (255 -\alpha) + c \times \alpha) \div 255
\]

Each of the `mlib_GraphicsDrawEllipse_A_*()` functions draws an ellipse with antialiasing and alpha blending.

**Parameters**  Each of the functions takes some of the following arguments:
- `buffer`  Pointer to the image into which the function is drawing.
- `x`  X coordinate of the center.
- `y`  Y coordinate of the center.
- `a`  Major semiaxis of the ellipse.
- `b`  Minor semiaxis of the ellipse.
- `t`  Angle of major semiaxis in radians.
- `c`  Color used in the drawing.
- `c2`  Alternation color.
- `alpha`  Alpha value for blending. \(0 \leq \alpha \leq 255\).

**Return Values**  Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_GraphicsDrawArc(3MLIB), mlib_GraphicsDrawCircle(3MLIB), attributes(5)`
Name  

mlib_GraphicsDrawLine_X_8, mlib_GraphicsDrawLine_X_32,
mlib_GraphicsDrawLine_A_8, mlib_GraphicsDrawLine_A_32,
mlib_GraphicsDrawLine_B_8, mlib_GraphicsDrawLine_B_32,
mlib_GraphicsDrawLine_G_8, mlib_GraphicsDrawLine_G_32,
mlib_GraphicsDrawLine_Z_8, mlib_GraphicsDrawLine_Z_32,
mlib_GraphicsDrawLine_AB_8, mlib_GraphicsDrawLine_AB_32,
mlib_GraphicsDrawLine_ABG_8, mlib_GraphicsDrawLine_ABG_32,
mlib_GraphicsDrawLine_ABGZ_8, mlib_GraphicsDrawLine_ABGZ_32,
mlib_GraphicsDrawLine_ABZ_8, mlib_GraphicsDrawLine_ABZ_32,
mlib_GraphicsDrawLine_AG_8, mlib_GraphicsDrawLine_AG_32,
mlib_GraphicsDrawLine_AGZ_8, mlib_GraphicsDrawLine_AGZ_32,
mlib_GraphicsDrawLine_AZ_8, mlib_GraphicsDrawLine_AZ_32,
mlib_GraphicsDrawLine_BG_8, mlib_GraphicsDrawLine_BG_32,
mlib_GraphicsDrawLine_BGZ_8, mlib_GraphicsDrawLine_BGZ_32,
mlib_GraphicsDrawLine_BZ_8, mlib_GraphicsDrawLine_BZ_32,
mlib_GraphicsDrawLine_GZ_8, mlib_GraphicsDrawLine_GZ_32 - draw line

Synopsis  

c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawLine_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c);
mlib_status mlib_GraphicsDrawLine_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c);
mlib_status mlib_GraphicsDrawLine_X_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawLine_X_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawLine_A_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c);
mlib_status mlib_GraphicsDrawLine_A_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c);
mlib_status mlib_GraphicsDrawLine_B_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLine_B_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLine_G_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c, mlib_s32 c1);
mlib_status mlib_GraphicsDrawLine_G_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s32 c1, mlib_s32 c2);
mlib_status mlib_GraphicsDrawLine_Z_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawLine_Z_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawLine_AB_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_AB_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABG_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABG_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_ABZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1,
    mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_AG_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2);

mlib_status mlib_GraphicsDrawLine_AG_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2);
mlib_status mlib_GraphicsDrawLine_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2);

mlib_status mlib_GraphicsDrawLine_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawLine_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawLine_BG_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2,
    mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_BG_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2,
    mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s32 c1, mlib_s32 c2, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_BZ_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_BZ_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLine_GZ_8(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2);

mlib_status mlib_GraphicsDrawLine_GZ_32(mlib_image *buffer,
    mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
    mlib_s32 c1, mlib_s32 c2);
Each of the `mlib_GraphicsDrawLine_*(x,y)` functions draws a line between `(x1,y1)` and `(x2,y2)`. Each of the `mlib_GraphicsDrawLine_X_*(x,y)` functions draws a line between `(x1,y1)` and `(x2,y2)` in Xor mode as follows:

```
data[x,y] ^= c ^ c2
```

Each of the `mlib_GraphicsDrawLine_A_*(x,y)` functions draws a line between `(x1,y1)` and `(x2,y2)` with antialiasing.

Each of the `mlib_GraphicsDrawLine_B_*(x,y)` functions draws a line between `(x1,y1)` and `(x2,y2)` with alpha blending as follows:

```
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
```

Each of the `mlib_GraphicsDrawLine_G_*(x,y)` functions draws a line between `(x1,y1)` and `(x2,y2)` with Gouraud shading.

Each of the `mlib_GraphicsDrawLine_Z_*(x,y)` functions draws a line between `(x1,y1)` and `(x2,y2)` with Z buffering.

Each of the other functions draws a line between `(x1,y1)` and `(x2,y2)` with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

### Parameters

Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `zbuffer` Pointer to the image that holds the Z buffer.
- `x1` X coordinate of the first point.
- `y1` Y coordinate of the first point.
- `z1` Z coordinate of the first point.
- `x2` X coordinate of the second point.
- `y2` Y coordinate of the second point.
- `z2` Z coordinate of the second point.
- `c` Color used in the drawing.
- `c1` Color of the first point.
- `c2` Color of the second point, or the alternation color in the Xor mode.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_GraphicsDrawLineSet(3MLIB), mlib_GraphicsDrawLineFanSet(3MLIB), mlib_GraphicsDrawLineStripSet(3MLIB), attributes(5)
mlib_GraphicsDrawLineFanSet(3MLIB)

Name
mlib_GraphicsDrawLineFanSet, mlib_GraphicsDrawLineFanSet_8,
mlib_GraphicsDrawLineFanSet_32, mlib_GraphicsDrawLineFanSet_X_8,
mlib_GraphicsDrawLineFanSet_X_32, mlib_GraphicsDrawLineFanSet_A_8,
mlib_GraphicsDrawLineFanSet_A_32, mlib_GraphicsDrawLineFanSet_B_8,
mlib_GraphicsDrawLineFanSet_B_32, mlib_GraphicsDrawLineFanSet_G_8,
mlib_GraphicsDrawLineFanSet_G_32, mlib_GraphicsDrawLineFanSet_Z_8,
mlib_GraphicsDrawLineFanSet_Z_32, mlib_GraphicsDrawLineFanSet_AB_8,
mlib_GraphicsDrawLineFanSet_AB_32, mlib_GraphicsDrawLineFanSet_ABG_8,
mlib_GraphicsDrawLineFanSet_ABG_32, mlib_GraphicsDrawLineFanSet_ABGZ_8,
mlib_GraphicsDrawLineFanSet_ABGZ_32, mlib_GraphicsDrawLineFanSet_ABZ_8,
mlib_GraphicsDrawLineFanSet_ABZ_32, mlib_GraphicsDrawLineFanSet_AG_8,
mlib_GraphicsDrawLineFanSet_AG_32, mlib_GraphicsDrawLineFanSet_AGZ_8,
mlib_GraphicsDrawLineFanSet_AGZ_32, mlib_GraphicsDrawLineFanSet_AZ_8,
mlib_GraphicsDrawLineFanSet_AZ_32, mlib_GraphicsDrawLineFanSet_BG_8,
mlib_GraphicsDrawLineFanSet_BG_32, mlib_GraphicsDrawLineFanSet_BGZ_8,
mlib_GraphicsDrawLineFanSet_BGZ_32, mlib_GraphicsDrawLineFanSet_BZ_8,
mlib_GraphicsDrawLineFanSet_BZ_32, mlib_GraphicsDrawLineFanSet_GZ_8,
mlib_GraphicsDrawLineFanSet_GZ_32 – draw line set where all members of the set have one
common end point

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_GraphicsDrawLineFanSet_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineFanSet_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineFanSet_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
mlib_status mlib_GraphicsDrawLineFanSet_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
mlib_status mlib_GraphicsDrawLineFanSet_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineFanSet_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineFanSet_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

Multimedia Library Functions - Part 1 49
mlib_status mlib_GraphicsDrawLineFanSet_G_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);
mlib_status mlib_GraphicsDrawLineFanSet_G_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);
mlib_status mlib_GraphicsDrawLineFanSet_Z_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineFanSet_Z_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineFanSet_AB_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_AB_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_ABG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_ABG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_ABGZ_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_ABGZ_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_ABZ_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_ABZ_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineFanSet_AG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);
mlib_status mlib_GraphicsDrawLineFanSet_AG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);
Each of the `mlib_GraphicsDrawLineFanSet_*()` functions draws a set of lines connecting 

\((x_1,y_1)\) with \((x_2,y_2)\), \((x_1,y_1)\) with \((x_3,y_3)\), ..., and \((x_1,y_1)\) with \((x_n,y_n)\).

Each of the `mlib_GraphicsDrawLineFanSet_X_*()` functions draws a set of lines in Xor mode as follows:
Each of the mlib_GraphicsDrawLineFanSet_A_*() functions draws a set of lines with antialiasing.

Each of the mlib_GraphicsDrawLineFanSet_B_*() functions draws a set of lines with alpha blending as follows:

\[
data[x,y] = (data[x,y] \times (255 - a) + c \times a) / 255\]

Each of the mlib_GraphicsDrawLineFanSet_G_*() functions draws a set of lines with Gouraud shading.

Each of the mlib_GraphicsDrawLineFanSet_Z_*() functions draws a set of lines with Z buffering.

Each of the other functions draws a set of lines with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

**Parameters**

Each of the functions takes some of the following arguments:

- **buffer**: Pointer to the image into which the function is drawing.
- **zbuffer**: Pointer to the image that holds the Z buffer.
- **x**: Pointer to array of X coordinates of the points.
- **y**: Pointer to array of Y coordinates of the points.
- **z**: Pointer to array of Z coordinates of the points.
- **npoints**: Number of points in the arrays.
- **c**: Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- **c2**: Alternation color.
- **a**: Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values**

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_GraphicsDrawLine(3MLIB), mlib_GraphicsDrawLineSet(3MLIB),
mlib_GraphicsDrawLineStripSet(3MLIB), attributes(5)
#Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib graphicsDrawLineSet_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_AB_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_AB_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_ABG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_ABG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_ABGZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_ABGZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_ABZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_ABZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_AG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_AGZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_AGZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib graphicsDrawLineSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_BGZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_BGZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a, mlib_s32 b);
mlib_status mlib graphicsDrawLineSet_BZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_BZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_GZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib graphicsDrawLineSet_GZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
```
mlib_status mlib_GraphicsDrawLineSet_G_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineSet_G_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineSet_Z_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_Z_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_AB_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_AB_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_ABG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_ABG_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_ABGZ_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_ABGZ_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_ABZ_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_ABZ_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawLineSet_AG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineSet_AGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineSet_AGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineSet_AZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_AZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineSet_BG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_BG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_BGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_BGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_BZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_BZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineSet_GZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineSet_GZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 npoints, const mlib_s32 *c);
Each of the `mlib_GraphicsDrawLineSet_*()` functions draws a set of lines connecting 
\((x_1, y_1)\) with \((x_2, y_2)\), \((x_3, y_3)\) with \((x_4, y_4)\), ..., and \((x_{n-1}, y_{n-1})\) with \((x_n, y_n)\).

Each of the `mlib_GraphicsDrawLineSet_X_*()` functions draws a set of lines in Xor mode as follows:

\[
data[x,y] \leftarrow c \oplus c^2
\]

Each of the `mlib_GraphicsDrawLineSet_A_*()` functions draws a set of lines with antialiasing.

Each of the `mlib_GraphicsDrawLineSet_B_*()` functions draws a set of lines with alpha blending as follows:

\[
data[x,y] = (data[x,y] \times (255 - a) + c \times a) / 255
\]

Each of the `mlib_GraphicsDrawLineSet_G_*()` functions draws a set of lines with Gouraud shading.

Each of the `mlib_GraphicsDrawLineSet_Z_*()` functions draws a set of lines with Z buffering.

Each of the other functions draws a set of lines with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

### Parameters

Each of the functions takes some of the following arguments:

- **buffer**: Pointer to the image into which the function is drawing.
- **zbuffer**: Pointer to the image that holds the Z buffer.
- **x**: Pointer to array of X coordinates of the points.
- **y**: Pointer to array of Y coordinates of the points.
- **z**: Pointer to array of Z coordinates of the points.
- **npoints**: Number of points in the arrays. `npoints` must be a multiple of 2.
- **c**: Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- **c2**: Alternation color.
- **a**: Alpha value for blending. \(0 \leq a \leq 255\).

### Return Values

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:
### mlib_GraphicsDrawLineSet(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_GraphicsDrawLine(3MLIB), mlib_GraphicsDrawLineFanSet(3MLIB), mlib_GraphicsDrawLineStripSet(3MLIB), attributes(5)
mlib_GraphicsDrawLineStripSet(3MLIB)

Name  
mlib_GraphicsDrawLineStripSet, mlib_GraphicsDrawLineStripSet_8,
mlib_GraphicsDrawLineStripSet_32, mlib_GraphicsDrawLineStripSet_X_8,
mlib_GraphicsDrawLineStripSet_X_32, mlib_GraphicsDrawLineStripSet_A_8,
mlib_GraphicsDrawLineStripSet_A_32, mlib_GraphicsDrawLineStripSet_B_8,
mlib_GraphicsDrawLineStripSet_B_32, mlib_GraphicsDrawLineStripSet_G_8,
mlib_GraphicsDrawLineStripSet_G_32, mlib_GraphicsDrawLineStripSet_Z_8,
mlib_GraphicsDrawLineStripSet_Z_32, mlib_GraphicsDrawLineStripSet_AB_8,
mlib_GraphicsDrawLineStripSet_AB_32, mlib_GraphicsDrawLineStripSet_ABG_8,
mlib_GraphicsDrawLineStripSet_ABG_32, mlib_GraphicsDrawLineStripSet_ABGZ_8,
mlib_GraphicsDrawLineStripSet_ABGZ_32, mlib_GraphicsDrawLineStripSet_ABZ_8,
mlib_GraphicsDrawLineStripSet_ABZ_32, mlib_GraphicsDrawLineStripSet_AG_8,
mlib_GraphicsDrawLineStripSet_AG_32, mlib_GraphicsDrawLineStripSet_AGZ_8,
mlib_GraphicsDrawLineStripSet_AGZ_32, mlib_GraphicsDrawLineStripSet_AZ_8,
mlib_GraphicsDrawLineStripSet_AZ_32, mlib_GraphicsDrawLineStripSet_BG_8,
mlib_GraphicsDrawLineStripSet_BG_32, mlib_GraphicsDrawLineStripSet_BGZ_8,
mlib_GraphicsDrawLineStripSet_BGZ_32, mlib_GraphicsDrawLineStripSet_BZ_8,
mlib_GraphicsDrawLineStripSet_BZ_32, mlib_GraphicsDrawLineStripSet_GZ_8,
mlib_GraphicsDrawLineStripSet_GZ_32 - draw line set where each member of the set starts
at the point where the previous member ended

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib GraphicsDrawLineStripSet_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib GraphicsDrawLineStripSet_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib GraphicsDrawLineStripSet_X_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 c2);
mlib_status mlib GraphicsDrawLineStripSet_X_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 c, mlib_s32 c2);
mlib_status mlib GraphicsDrawLineStripSet_A_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib GraphicsDrawLineStripSet_A_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib GraphicsDrawLineStripSet_B_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib GraphicsDrawLineStripSet_B_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawLineStripSet_G_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineStripSet_G_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawLineStripSet_Z_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineStripSet_Z_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z,
mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawLineStripSet_AB_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c,
mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_AB_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABG_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABGZ_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABGZ_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABZ_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c,
mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_ABZ_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawLineStripSet_AG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);
mlib_GraphicsDrawLineStripSet_AG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_GraphicsDrawLineStripSet_AGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_GraphicsDrawLineStripSet_AGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_GraphicsDrawLineStripSet_AGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_GraphicsDrawLineStripSet_AZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_GraphicsDrawLineStripSet_AZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_GraphicsDrawLineStripSet_BG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_GraphicsDrawLineStripSet_BG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_GraphicsDrawLineStripSet_BGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_GraphicsDrawLineStripSet_BGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_GraphicsDrawLineStripSet_BZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_GraphicsDrawLineStripSet_BZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_GraphicsDrawLineStripSet_GZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_GraphicsDrawLineStripSet_GZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
Each of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_\ast()} functions draws a set of lines connecting \((x_1, y_1)\) with \((x_2, y_2)\), \((x_2, y_2)\) with \((x_3, y_3)\), ..., and \((x_{n-1}, y_{n-1})\) with \((x_n, y_n)\).

Each of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_X\_\ast()} functions draws a set of lines in Xor mode as follows:

\[
data[x,y] ^= c ^ c2
\]

Each of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_A\_\ast()} functions draws a set of lines with antialiasing.

Each of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_B\_\ast()} functions draws a set of lines with alpha blending as follows:

\[
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
\]

Each of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_G\_\ast()} functions draws a set of lines with Gouraud shading.

Each of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_Z\_\ast()} functions draws a set of lines with Z buffering.

Each of the other functions draws a set of lines with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Note that the \texttt{mlib\_Graphics\_Draw\_Polyline\_\ast()} functions are aliases of the \texttt{mlib\_Graphics\_Draw\_Line\_Strip\_Set\_\ast()} functions.

Each of the functions takes some of the following arguments:

- \texttt{buffer} Pointer to the image into which the function is drawing.
- \texttt{zbuffer} Pointer to the image that holds the Z buffer.
- \texttt{x} Pointer to array of X coordinates of the points.
- \texttt{y} Pointer to array of Y coordinates of the points.
- \texttt{z} Pointer to array of Z coordinates of the points.
- \texttt{npoints} Number of points in the arrays.
- \texttt{c} Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- \texttt{c2} Alternation color.
- \texttt{a} Alpha value for blending. \(0 \leq a \leq 255\).

Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_GraphicsDrawLine(3MLIB), mlib_GraphicsDrawLineSet(3MLIB),
mlib_GraphicsDrawLineFanSet(3MLIB), mlib_GraphicsDrawPolyline(3MLIB),
attributes(5)
Name  
mlib_GraphicsDrawPoint, mlib_GraphicsDrawPoint_8, mlib_GraphicsDrawPoint_32,
mlib_GraphicsDrawPoint_X_8, mlib_GraphicsDrawPoint_X_32,
mlib_GraphicsDrawPoint_B_8, mlib_GraphicsDrawPoint_B_32 – draw point

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawPoint_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c);
mlib_status mlib_GraphicsDrawPoint_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c);
mlib_status mlib_GraphicsDrawPoint_X_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawPoint_X_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawPoint_B_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawPoint_B_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 c, mlib_s32 a);

Description  
Each of the mlib_GraphicsDrawPoint_*() functions draws a point at \((x,y)\) in color \(c\).

Each of the mlib_GraphicsDrawPoint_X_*() functions draws a point at \((x,y)\) in Xor mode as follows:

\[ data[x,y] \oplus c^c 2 \]

Each of the mlib_GraphicsDrawPoint_B_*() functions draws a point at \((x,y)\) with alpha blending as follows:

\[ data[x,y] = (data[x,y] \times (255 - a) + c \times a) / 255 \]

Parameters  
Each of the functions takes some of the following arguments:

- **buffer**  
  Pointer to the image into which the function is drawing.

- **x**  
  X coordinate of the point.

- **y**  
  Y coordinate of the point.

- **c**  
  Color used in the drawing.

- **c2**  
  Alternation color.

- **a**  
  Alpha value for blending, \(0 \leq a \leq 255\).
Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_GraphicsDrawPointSet(3MLIB), attributes(5)

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawPointSet_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawPointSet_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawPointSet_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawPointSet_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsDrawPointSet_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawPointSet_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawPolypoint_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawPolypoint_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawPolypoint_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
mlib_status mlib_GraphicsDrawPolypoint_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
mlib_status mlib_GraphicsDrawPolypoint_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c);
mlib_status mlib_GraphicsDrawPolypoint_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
Each of the `mlib_GraphicsDrawPointSet_*( )` and `mlib_GraphicsDrawPolypoint_*( )` functions draws a set of points at \((x_1, y_1), (x_2, y_2), \ldots, \text{and} \ (x_n, y_n)\).

Each of the `mlib_GraphicsDrawPointSet_X_*( )` and `mlib_GraphicsDrawPolypoint_X_*( )` functions draws a set of points at \((x_1, y_1), (x_2, y_2), \ldots, \text{and} \ (x_n, y_n)\) in Xor mode as follows:

\[
data[x, y] \ XOR \ c \ ^ \ 2
\]

Each of the `mlib_GraphicsDrawPointSet_B_*( )` and `mlib_GraphicsDrawPolypoint_B_*( )` functions draws a set of points at \((x_1, y_1), (x_2, y_2), \ldots, \text{and} \ (x_n, y_n)\) with alpha blending as follows:

\[
data[x, y] = \frac{(data[x, y] \times (255 - a) + c \times a)}{255}
\]

The `mlib_GraphicsDrawPolypoint_*( )` functions are aliases of the `mlib_GraphicsDrawPointSet_*( )` functions.

Parameters
Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `x` Pointer to array of X coordinates of the points.
- `y` Pointer to array of Y coordinates of the points.
- `npoints` Number of points in the arrays.
- `c` Color used in the drawing.
- `c2` Alternation color.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).

Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_GraphicsDrawPoint(3MLIB), attributes(5)`
Name
mlib_GraphicsDrawPolygon, mlib_GraphicsDrawPolygon_8,
mlib_GraphicsDrawPolygon_32, mlib_GraphicsDrawPolygon_X_8,
mlib_GraphicsDrawPolygon_X_32, mlib_GraphicsDrawPolygon_A_8,
mlib_GraphicsDrawPolygon_A_32, mlib_GraphicsDrawPolygon_B_8,
mlib_GraphicsDrawPolygon_B_32, mlib_GraphicsDrawPolygon_G_8,
mlib_GraphicsDrawPolygon_G_32, mlib_GraphicsDrawPolygon_Z_8,
mlib_GraphicsDrawPolygon_Z_32, mlib_GraphicsDrawPolygon_AB_8,
mlib_GraphicsDrawPolygon_AB_32, mlib_GraphicsDrawPolygon_ABG_8,
mlib_GraphicsDrawPolygon_ABG_32, mlib_GraphicsDrawPolygon_ABGZ_8,
mlib_GraphicsDrawPolygon_ABGZ_32, mlib_GraphicsDrawPolygon_ABZ_8,
mlib_GraphicsDrawPolygon_ABZ_32, mlib_GraphicsDrawPolygon_AG_8,
mlib_GraphicsDrawPolygon_AG_32, mlib_GraphicsDrawPolygon_AGZ_8,
mlib_GraphicsDrawPolygon_AGZ_32, mlib_GraphicsDrawPolygon_AZ_8,
mlib_GraphicsDrawPolygon_AZ_32, mlib_GraphicsDrawPolygon_BG_8,
mlib_GraphicsDrawPolygon_BG_32, mlib_GraphicsDrawPolygon_BGZ_8,
mlib_GraphicsDrawPolygon_BGZ_32, mlib_GraphicsDrawPolygon_BZ_8,
mlib_GraphicsDrawPolygon_BZ_32, mlib_GraphicsDrawPolygon_GZ_8,
mlib_GraphicsDrawPolygon_GZ_32 – draw polygon

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawPolygon_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mlib_status mlib_GraphicsDrawPolygon_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);

mlib_status mlib_GraphicsDrawPolygon_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
mlib_status mlib_GraphicsDrawPolygon_G_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolygon_G_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolygon_Z_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *z, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_Z_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *z, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_AB_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_AB_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_ABG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_ABG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_ABGZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *z, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_ABGZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *z, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_ABZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_ABZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_AG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c);
mlib_status mlib_GraphicsDrawPolygon_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolygon_AG_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolygon_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolygon_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    mlib_s32 c);

mlib_status mlib_GraphicsDrawPolygon_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolygon_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s32 npoints,
    const mlib_s32 *c);
mlib_status mlib_GraphicsDrawPolygon_GZ_32(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    const mlib_s32 *c);

**Description** Each of the mlib_GraphicsDrawPolygon_*() functions draws a polygon enclosing \((x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\).

Each of the mlib_GraphicsDrawPolygon_X_*() functions draws a polygon in Xor mode as follows:

\[
data[x,y] ^= c ^ c2
\]

Each of the mlib_GraphicsDrawPolygon_A_*() functions draws a polygon with antialiasing.

Each of the mlib_GraphicsDrawPolygon_B_*() functions draws a polygon with alpha blending as follows:

\[
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
\]

Each of the mlib_GraphicsDrawPolygon_G_*() functions draws a polygon with Gouraud shading.

Each of the mlib_GraphicsDrawPolygon_Z_*() functions draws a polygon with Z buffering.

Each of the other functions draws a polygon with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

**Parameters** Each of the functions takes some of the following arguments:

- **buffer** Pointer to the image into which the function is drawing.
- **zbuffer** Pointer to the image that holds the Z buffer.
- **x** Pointer to the array of X coordinates of the vertices.
- **y** Pointer to the array of Y coordinates of the vertices.
- **z** Pointer to the array of Z coordinates of the vertices.
- **npoints** Number of vertices in the arrays.
- **c** Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- **c2** Alternation color.
- **a** Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_GraphicsFillPolygon(3MLIB), attributes(5)
**Name**

mlib_GraphicsDrawPolyline, mlib_GraphicsDrawPolyline_8,
mlib_GraphicsDrawPolyline_32, mlib_GraphicsDrawPolyline_X_8,
mlib_GraphicsDrawPolyline_X_32, mlib_GraphicsDrawPolyline_A_8,
mlib_GraphicsDrawPolyline_A_32, mlib_GraphicsDrawPolyline_B_8,
mlib_GraphicsDrawPolyline_B_32, mlib_GraphicsDrawPolyline_G_8,
mlib_GraphicsDrawPolyline_G_32, mlib_GraphicsDrawPolyline_Z_8,
mlib_GraphicsDrawPolyline_Z_32, mlib_GraphicsDrawPolyline_AB_8,
mlib_GraphicsDrawPolyline_AB_32, mlib_GraphicsDrawPolyline_ABG_8,
mlib_GraphicsDrawPolyline_ABG_32, mlib_GraphicsDrawPolyline_ABGZ_8,
mlib_GraphicsDrawPolyline_ABGZ_32, mlib_GraphicsDrawPolyline_ABZ_8,
mlib_GraphicsDrawPolyline_ABZ_32, mlib_GraphicsDrawPolyline_AG_8,
mlib_GraphicsDrawPolyline_AG_32, mlib_GraphicsDrawPolyline_AGZ_8,
mlib_GraphicsDrawPolyline_AGZ_32, mlib_GraphicsDrawPolyline_AZ_8,
mlib_GraphicsDrawPolyline_AZ_32, mlib_GraphicsDrawPolyline_BG_8,
mlib_GraphicsDrawPolyline_BG_32, mlib_GraphicsDrawPolyline_BGZ_8,
mlib_GraphicsDrawPolyline_BGZ_32, mlib_GraphicsDrawPolyline_BZ_8,
mlib_GraphicsDrawPolyline_BZ_32, mlib_GraphicsDrawPolyline_GZ_8,
mlib_GraphicsDrawPolyline_GZ_32 – draw polyline

**Synopsis**

cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawPolyline_8(mlib_image *buffer,
                                      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_32(mlib_image *buffer,
                                         const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_X_8(mlib_image *buffer,
                                          const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_X_32(mlib_image *buffer,
                                          const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsDrawPolyline_A_8(mlib_image *buffer,
                                          const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_A_32(mlib_image *buffer,
                                          const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_B_8(mlib_image *buffer,
                                          const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_B_32(mlib_image *buffer,
                                          const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

Multimedia Library Functions - Part 1 73
mlib_status mlib_GraphicsDrawPolyline_G_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolyline_G_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawPolyline_Z_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_Z_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawPolyline_AB_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_AB_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_ABG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_ABG_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_ABGZ_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_ABGZ_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_ABZ_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_ABZ_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_AG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawPolyline_AG_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);
Each of the `mlib_GraphicsDrawPolyline_AGZ_8()` functions draws a polyline connecting 
\((x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\).

Each of the `mlib_GraphicsDrawPolyline_AGZ_32()` functions draws a polyline in Xor mode as follows:
Each of the `mlib_GraphicsDrawPolyline_A_*()` functions draws a polyline with antialiasing.

Each of the `mlib_GraphicsDrawPolyline_B_*()` functions draws a polyline with alpha blending as follows:

\[ \text{data}[x,y] = \frac{\text{data}[x,y] \cdot (255 - \alpha) + \alpha \cdot c}{255} \]

Each of the `mlib_GraphicsDrawPolyline_G_*()` functions draws a polyline with Gouraud shading.

Each of the `mlib_GraphicsDrawPolyline_Z_*()` functions draws a polyline with Z buffering.

Each of the other functions draws a polyline with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Note that the `mlib_GraphicsDrawPolyline_*()` functions are aliases of the `mlib_GraphicsDrawLineStripSet_*()` functions.

**Parameters**  
Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `zbuffer` Pointer to the image that holds the Z buffer.
- `x` Pointer to array of X coordinates of the points.
- `y` Pointer to array of Y coordinates of the points.
- `z` Pointer to array of Z coordinates of the points.
- `npoints` Number of points in the arrays.
- `c` Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- `c2` Alternation color.
- `a` Alpha value for blending. \(0 \leq \alpha \leq 255\).

**Return Values**  
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

mlib_GraphicsDrawLine(3MLIB), mlib_GraphicsDrawLineSet(3MLIB),
mlib_GraphicsDrawLineFanSet(3MLIB), mlib_GraphicsDrawLineStripSet(3MLIB),
attributes(5)
Each of the `mlib_GraphicsDrawRectangle_*()` functions draws a rectangle with the upper-left corner at \((x, y)\), width \(w\), and height \(h\).

Each of the `mlib_GraphicsDrawRectangle_X_*( )` functions draws a rectangle in Xor mode as follows:

\[
data[x, y] ^= c ^ c2
\]

Each of the `mlib_GraphicsDrawRectangle_B_*( )` functions draws a rectangle with alpha blending as follows:

\[
data[x, y] = (data[x, y] * (255 - a) + c * a) / 255
\]

### Parameters

- **buffer** Pointer to the image into which the function is drawing.
- **x** X coordinate of the upper-left corner of the rectangle.
- **y** Y coordinate of the upper-left corner of the rectangle.
- **w** Width of the rectangle.
- **h** Height of the rectangle.
- **c** Color used in the drawing.
c2 Alternation color.

a Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_GraphicsFillRectangle(3MLIB), attributes(5)
Name
mlib_GraphicsDrawTriangle, mlib_GraphicsDrawTriangle_8,
mlib_GraphicsDrawTriangle_32, mlib_GraphicsDrawTriangle_X_8,
mlib_GraphicsDrawTriangle_X_32, mlib_GraphicsDrawTriangle_A_8,
mlib_GraphicsDrawTriangle_A_32, mlib_GraphicsDrawTriangle_B_8,
mlib_GraphicsDrawTriangle_B_32, mlib_GraphicsDrawTriangle_G_8,
mlib_GraphicsDrawTriangle_G_32, mlib_GraphicsDrawTriangle_Z_8,
mlib_GraphicsDrawTriangle_Z_32, mlib_GraphicsDrawTriangle_AB_8,
mlib_GraphicsDrawTriangle_AB_32, mlib_GraphicsDrawTriangle_ABG_8,
mlib_GraphicsDrawTriangle_ABG_32, mlib_GraphicsDrawTriangle_ABGZ_8,
mlib_GraphicsDrawTriangle_ABGZ_32, mlib_GraphicsDrawTriangle_ABZ_8,
mlib_GraphicsDrawTriangle_ABZ_32, mlib_GraphicsDrawTriangle_AG_8,
mlib_GraphicsDrawTriangle_AG_32, mlib_GraphicsDrawTriangle_AGZ_8,
mlib_GraphicsDrawTriangle_AGZ_32, mlib_GraphicsDrawTriangle_AZ_8,
mlib_GraphicsDrawTriangle_AZ_32, mlib_GraphicsDrawTriangle_BG_8,
mlib_GraphicsDrawTriangle_BG_32, mlib_GraphicsDrawTriangle_BGZ_8,
mlib_GraphicsDrawTriangle_BGZ_32, mlib_GraphicsDrawTriangle_BZ_8,
mlib_GraphicsDrawTriangle_BZ_32, mlib_GraphicsDrawTriangle_GZ_8,
mlib_GraphicsDrawTriangle_GZ_32 – draw triangle

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawTriangle_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_X_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsDrawTriangle_X_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsDrawTriangle_A_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_A_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_B_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_B_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangle_B_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_G_8(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_G_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_Z_8(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_Z_32(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_AB_8(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_AB_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_ABG_8(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_ABG_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_ABGZ_8(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_ABGZ_32(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_ABZ_8(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangle_ABZ_32(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_AG_8(mlib_image *buffer,
  mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
  mlib_s16 x3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_AG_32(mlib_image *buffer,
  mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
  mlib_s16 x3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_AGZ_8(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_AGZ_32(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_AZ_8(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_AZ_32(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangle_BG_8(mlib_image *buffer,
  mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
  mlib_s16 x3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_BG_32(mlib_image *buffer,
  mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2,
  mlib_s16 x3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_BGZ_8(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangle_BGZ_32(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c1,
  mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_BZ_8(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
  mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_BZ_32(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c,
  mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangle_GZ_8(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
  mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsDrawTriangle_GZ_32(mlib_image *buffer,
  mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
  mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
  mlib_s16 z3, mlib_s32 c1,
  mlib_s32 c2, mlib_s32 c3);

**Description**

Each of the mlib_GraphicsDrawTriangle_*() functions draws a triangle with the vertices at 
(x1,y1), (x2,y2), and (x3,y3).

Each of the mlib_GraphicsDrawTriangle_X_*() functions draws a triangle in Xor mode as follows:

\[ \text{data}[x,y] \oplus c \quad c2 \]

Each of the mlib_GraphicsDrawTriangle_A_*() functions draws a triangle with antialiasing.

Each of the mlib_GraphicsDrawTriangle_B_*() functions draws a triangle with alpha blending as follows:

\[ \text{data}[x,y] = (\text{data}[x,y] \times (255 - a) + c \times a) / 255 \]

Each of the mlib_GraphicsDrawTriangle_G_*() functions draws a triangle with Gouraud shading.

Each of the mlib_GraphicsDrawTriangle_Z_*() functions draws a triangle with Z buffering.

Each of the other functions draws a triangle with a combination of two or more features like 
antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).
Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `zbuffer` Pointer to the image that holds the Z buffer.
- `x1` X coordinate of the first vertex.
- `y1` Y coordinate of the first vertex.
- `z1` Z coordinate of the first vertex.
- `x2` X coordinate of the second vertex.
- `y2` Y coordinate of the second vertex.
- `z2` Z coordinate of the second vertex.
- `x3` X coordinate of the third vertex.
- `y3` Y coordinate of the third vertex.
- `z3` Z coordinate of the third vertex.
- `c` Color used in the drawing.
- `c1` Color of the first vertex.
- `c2` Color of the second vertex, or the alternation color in Xor Mode.
- `c3` Color of the third vertex.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes\(\text{(5)}\) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
`mlib_GraphicsFillTriangle\(3\text{MLIB})`, attributes\(\text{(5)}\)
Name  
mlib_GraphicsDrawTriangleFanSet, mlib_GraphicsDrawTriangleFanSet_8,
mlib_GraphicsDrawTriangleFanSet_32, mlib_GraphicsDrawTriangleFanSet_X_8,
mlib_GraphicsDrawTriangleFanSet_X_32, mlib_GraphicsDrawTriangleFanSet_A_8,
mlib_GraphicsDrawTriangleFanSet_A_32, mlib_GraphicsDrawTriangleFanSet_B_8,
mlib_GraphicsDrawTriangleFanSet_B_32, mlib_GraphicsDrawTriangleFanSet_G_8,
mlib_GraphicsDrawTriangleFanSet_G_32, mlib_GraphicsDrawTriangleFanSet_Z_8,
mlib_GraphicsDrawTriangleFanSet_Z_32, mlib_GraphicsDrawTriangleFanSet_AB_8,
mlib_GraphicsDrawTriangleFanSet_AB_32, mlib_GraphicsDrawTriangleFanSet_ABG_8,
mlib_GraphicsDrawTriangleFanSet_ABG_32,
mlib_GraphicsDrawTriangleFanSet_ABGZ_8,
mlib_GraphicsDrawTriangleFanSet_ABGZ_32, mlib_GraphicsDrawTriangleFanSet_ABZ_8,
mlib_GraphicsDrawTriangleFanSet_ABZ_32, mlib_GraphicsDrawTriangleFanSet_AG_8,
mlib_GraphicsDrawTriangleFanSet_AG_32, mlib_GraphicsDrawTriangleFanSet_AGZ_8,
mlib_GraphicsDrawTriangleFanSet_AGZ_32, mlib_GraphicsDrawTriangleFanSet_AZ_8,
mlib_GraphicsDrawTriangleFanSet_AZ_32, mlib_GraphicsDrawTriangleFanSet_BG_8,
mlib_GraphicsDrawTriangleFanSet_BG_32, mlib_GraphicsDrawTriangleFanSet_BGZ_8,
mlib_GraphicsDrawTriangleFanSet_BGZ_32, mlib_GraphicsDrawTriangleFanSet_BZ_8,
mlib_GraphicsDrawTriangleFanSet_BZ_32, mlib_GraphicsDrawTriangleFanSet_GZ_8,
mlib_GraphicsDrawTriangleFanSet_GZ_32 – draw triangle set where all members of the set
have a common vertex

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawTriangleFanSet_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleFanSet_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleFanSet_X_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);
mlib_status mlib_GraphicsDrawTriangleFanSet_X_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 c2);
mlib_status mlib_GraphicsDrawTriangleFanSet_A_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleFanSet_A_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleFanSet_B_8(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangleFanSet_B_32(mlib_image *buffer,
      const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
      mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangleFanSet_G_8(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleFanSet_G_32(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleFanSet_Z_8(mlib_image *buffer,  
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,  
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_Z_32(mlib_image *buffer,  
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,  
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AB_8(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AB_32(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_ABG_8(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_ABG_32(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_ABGZ_8(mlib_image *buffer,  
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,  
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_ABGZ_32(mlib_image *buffer,  
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,  
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_ABZ_8(mlib_image *buffer,  
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,  
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_ABZ_32(mlib_image *buffer,  
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,  
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AG_8(mlib_image *buffer,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,  
    mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangleFanSet_AG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_AZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleFanSet_BG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_BG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_BGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_BGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_BZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_BZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleFanSet_GZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleFanSet_GZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

Multimedia Library Functions - Part 1 87
Each of the `mlib_GraphicsDrawTriangleFanSet_*()` functions draws a set of triangles with vertices at \(\{(x_1,y_1), (x_2,y_2), (x_3,y_3)\}, \{(x_1,y_1), (x_3,y_3), (x_4,y_4)\}, \ldots\), and \(\{(x_1,y_1), (x_{n-1},y_{n-1}), (x_n,y_n)\}\).

Each of the `mlib_GraphicsDrawTriangleFanSet_X_*()` functions draws a set of triangles in Xor mode as follows:

\[
data[x,y] ^= c \wedge c2\]

Each of the `mlib_GraphicsDrawTriangleFanSet_A_*()` functions draws a set of triangles with antialiasing.

Each of the `mlib_GraphicsDrawTriangleFanSet_B_*()` functions draws a set of triangles with alpha blending as follows:

\[
data[x,y] = (data[x,y] \times (255 - a) + c \times a) / 255\]

Each of the `mlib_GraphicsDrawTriangleFanSet_G_*()` functions draws a set of triangles with Gouraud shading.

Each of the `mlib_GraphicsDrawTriangleFanSet_Z_*()` functions draws a set of triangles with Z buffering.

Each of the other functions draws a set of triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `zbuffer` Pointer to the image that holds the Z buffer.
- `x` Pointer to array of X coordinates of the points.
- `y` Pointer to array of Y coordinates of the points.
- `z` Pointer to array of Z coordinates of the points.
- `npoints` Number of points in the arrays.
- `c` Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- `c2` Alternation color.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:
mlib_GraphicsDrawTriangleFanSet(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_GraphicsDrawTriangle(3MLIB), mlib_GraphicsDrawTriangleSet(3MLIB), mlib_GraphicsDrawTriangleStripSet(3MLIB), attributes(5)
Name  mlib_GraphicsDrawTriangleSet, mlib_GraphicsDrawTriangleSet_8,
       mlib_GraphicsDrawTriangleSet_32, mlib_GraphicsDrawTriangleSet_X_8,
       mlib_GraphicsDrawTriangleSet_X_32, mlib_GraphicsDrawTriangleSet_A_8,
       mlib_GraphicsDrawTriangleSet_A_32, mlib_GraphicsDrawTriangleSet_B_8,
       mlib_GraphicsDrawTriangleSet_B_32, mlib_GraphicsDrawTriangleSet_G_8,
       mlib_GraphicsDrawTriangleSet_G_32, mlib_GraphicsDrawTriangleSet_Z_8,
       mlib_GraphicsDrawTriangleSet_Z_32, mlib_GraphicsDrawTriangleSet_AB_8,
       mlib_GraphicsDrawTriangleSet_AB_32, mlib_GraphicsDrawTriangleSet_ABG_8,
       mlib_GraphicsDrawTriangleSet_ABG_32, mlib_GraphicsDrawTriangleSet_ABGZ_8,
       mlib_GraphicsDrawTriangleSet_ABGZ_32, mlib_GraphicsDrawTriangleSet_ABZ_8,
       mlib_GraphicsDrawTriangleSet_ABZ_32, mlib_GraphicsDrawTriangleSet_AG_8,
       mlib_GraphicsDrawTriangleSet_AG_32, mlib_GraphicsDrawTriangleSet_AGZ_8,
       mlib_GraphicsDrawTriangleSet_AGZ_32, mlib_GraphicsDrawTriangleSet_AZ_8,
       mlib_GraphicsDrawTriangleSet_AZ_32, mlib_GraphicsDrawTriangleSet_BG_8,
       mlib_GraphicsDrawTriangleSet_BG_32, mlib_GraphicsDrawTriangleSet_BGZ_8,
       mlib_GraphicsDrawTriangleSet_BGZ_32, mlib_GraphicsDrawTriangleSet_BZ_8,
       mlib_GraphicsDrawTriangleSet_BZ_32, mlib_GraphicsDrawTriangleSet_GZ_8,
       mlib_GraphicsDrawTriangleSet_GZ_32 – draw triangle set where each member can have
       different vertices

Synopsis  cc [ flag...] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawTriangleSet_8(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleSet_32(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleSet_X_8(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsDrawTriangleSet_X_32(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsDrawTriangleSet_A_8(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleSet_A_32(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleSet_B_8(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_B_32(mlib_image *buffer,
       const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangleSet_G_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleSet_G_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleSet_Z_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleSet_Z_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleSet_AB_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_AB_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_ABG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_ABG_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_ABGZ_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_ABGZ_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_ABZ_8(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_ABZ_32(mlib_image *buffer,
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleSet_AG_8(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleSet_AG_32(mlib_image *buffer,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
const mlib_s32 *c);
Each of the `mlib_GraphicsDrawTriangleSet_8()` functions draws a set of triangles with vertices at \{ (x1,y1), (x2,y2), (x3,y3) \}, \{ (x4,y4), (x5,y5), (x6,y6) \}, ..., and \{ (xn-2,yn-2), (xn-1,yn-1), (xn,yn) \}.

Each of the `mlib_GraphicsDrawTriangleSet_X_8()` functions draws a set of triangles in Xor mode as follows:
Each of the `mlib_GraphicsDrawTriangleSet_A_*()` functions draws a set of triangles with antialiasing.

Each of the `mlib_GraphicsDrawTriangleSet_B_*()` functions draws a set of triangles with alpha blending as follows:

\[ \text{data}[x,y] = \frac{(\text{data}[x,y] \times (255 - a) + c \times a)}{255} \]

Each of the `mlib_GraphicsDrawTriangleSet_G_*()` functions draws a set of triangles with Gouraud shading.

Each of the `mlib_GraphicsDrawTriangleSet_Z_*()` functions draws a set of triangles with Z buffering.

Each of the other functions draws a set of triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

**Parameters**

Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `zbuffer` Pointer to the image that holds the Z buffer.
- `x` Pointer to array of X coordinates of the points.
- `y` Pointer to array of Y coordinates of the points.
- `z` Pointer to array of Z coordinates of the points.
- `npoints` Number of points in the arrays. `npoints` must be a multiple of 3.
- `c` Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- `c2` Alternation color.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_GraphicsDrawTriangle(3MLIB), mlib_GraphicsDrawTriangleFanSet(3MLIB),
mlib_GraphicsDrawTriangleStripSet(3MLIB), attributes(5)
Name  mlib_GraphicsDrawTriangleStripSet, mlib_GraphicsDrawTriangleStripSet_8,
mlib_GraphicsDrawTriangleStripSet_32, mlib_GraphicsDrawTriangleStripSet_X_8,
mlib_GraphicsDrawTriangleStripSet_X_32, mlib_GraphicsDrawTriangleStripSet_A_8,
mlib_GraphicsDrawTriangleStripSet_A_32, mlib_GraphicsDrawTriangleStripSet_B_8,
mlib_GraphicsDrawTriangleStripSet_G_8, mlib_GraphicsDrawTriangleStripSet_Z_8,
mlib_GraphicsDrawTriangleStripSet_AB_8, mlib_GraphicsDrawTriangleStripSet_ABG_8,
mlib_GraphicsDrawTriangleStripSet_ABGZ_8, mlib_GraphicsDrawTriangleStripSet_ABZ_8,
mlib_GraphicsDrawTriangleStripSet_AG_8, mlib_GraphicsDrawTriangleStripSet_AGZ_8,
mlib_GraphicsDrawTriangleStripSet_AZ_8, mlib_GraphicsDrawTriangleStripSet_BG_8,
mlib_GraphicsDrawTriangleStripSet_BGZ_8, mlib_GraphicsDrawTriangleStripSet_BZ_8,
mlib_GraphicsDrawTriangleStripSet_GZ_8–draw triangle set where the first side of each
member is common to the second side of the previous member

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsDrawTriangleStripSet_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleStripSet_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleStripSet_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
mlib_status mlib_GraphicsDrawTriangleStripSet_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 c2);
mlib_status mlib_GraphicsDrawTriangleStripSet_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleStripSet_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsDrawTriangleStripSet_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
mlib_status mlib_GraphicsDrawTriangleStripSet_B_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_G_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_G_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_Z_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleStripSet_Z_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AB_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_AB_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_ABG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_ABG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_ABGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_ABGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_ABZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_ABZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_AG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s32 *c);

mlib_GraphicsDrawTriangleStripSet(3MLIB)
mlib_status mlib_GraphicsDrawTriangleStripSet_AG_32(mlib_image *buffer, 
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AGZ_8(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AG_32(mlib_image *buffer, 
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AGZ_8(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AGZ_32(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AZ_8(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_AZ_32(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_BG_8(mlib_image *buffer, 
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_BG_32(mlib_image *buffer, 
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_BGZ_8(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_BGZ_32(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_BZ_8(mlib_image *buffer, 
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_BZ_32(mlib_image *buffer, 
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsDrawTriangleStripSet_GZ_8(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsDrawTriangleStripSet_GZ_32(mlib_image *buffer, 
mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
Each of the `mlib_GraphicsDrawTriangleStripSet_*()` functions draws a set of triangles with vertices at \((x_1, y_1), (x_2, y_2), (x_3, y_3)\), \((x_2, y_2), (x_3, y_3), (x_4, y_4)\), ..., and \((x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\).

Each of the `mlib_GraphicsDrawTriangleStripSet_X_*()` functions draws a set of triangles in Xor mode as follows:

```c
data[x,y] ^= c ^ c2
```

Each of the `mlib_GraphicsDrawTriangleStripSet_A_*()` functions draws a set of triangles with antialiasing.

Each of the `mlib_GraphicsDrawTriangleStripSet_B_*()` functions draws a set of triangles with alpha blending as follows:

```c
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
```

Each of the `mlib_GraphicsDrawTriangleStripSet_G_*()` functions draws a set of triangles with Gouraud shading.

Each of the `mlib_GraphicsDrawTriangleStripSet_Z_*()` functions draws a set of triangles with Z buffering.

Each of the other functions draws a set of triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

### Parameters
Each of the functions takes some of the following arguments:

- **buffer**: Pointer to the image into which the function is drawing.
- **zbuffer**: Pointer to the image that holds the Z buffer.
- **x**: Pointer to array of X coordinates of the points.
- **y**: Pointer to array of Y coordinates of the points.
- **z**: Pointer to array of Z coordinates of the points.
- **npoints**: Number of points in the arrays.
- **c**: Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- **c2**: Alternation color.
- **a**: Alpha value for blending. \(0 \leq a \leq 255\).

### Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:
mlib_GraphicsDrawTriangleStripSet(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_GraphicsDrawTriangle(3MLIB), mlib_GraphicsDrawTriangleSet(3MLIB),
mlib_GraphicsDrawTriangleFanSet(3MLIB), attributes(5)
Each of the \texttt{mlib\_GraphicsFillArc\_\*()} functions draws a filled arc with the center at \((x, y)\), radius \(r\), start angle \(t_1\), and end angle \(t_2\).

Each of the \texttt{mlib\_GraphicsFillArc\_X\_\*()} functions draws a filled arc in Xor mode as follows:

\[
data[x,y] ^= c^c_2
\]

Each of the \texttt{mlib\_GraphicsFillArc\_A\_\*()} functions draws a filled arc with antialiasing.
Each of the `mlib_GraphicsFillArc_B_*()` functions draws a filled arc with alpha blending as follows:

\[
data[x,y] = (data[x,y] \times (255 - a) + c \times a) / 255
\]

Each of the `mlib_GraphicsFillArc_AB_*()` functions draws a filled arc with antialiasing and alpha blending.

**Parameters** Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `x` X coordinate of the center.
- `y` Y coordinate of the center.
- `r` Radius of the arc.
- `t1` Start angle of the arc in radians.
- `t2` End angle of the arc in radians.
- `c` Color used in the drawing.
- `c2` Alternation color.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_GraphicsFillCircle(3MLIB), mlib_GraphicsFillEllipse(3MLIB), attributes(5)`

cc [ flag... ] file... -lmlib [ library... ] #include <mlib.h>

mlib_status mlib_GraphicsFillCircle_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsFillCircle_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsFillCircle_X_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFillCircle_X_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFillCircle_A_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsFillCircle_A_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c);
mlib_status mlib_GraphicsFillCircle_B_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsFillCircle_B_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsFillCircle_AB_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsFillCircle_AB_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 r, mlib_s32 c, mlib_s32 a);

Each of the mlib_GraphicsFillCircle_*() functions draws a filled circle with the center at \((x, y)\) and radius \(r\).

Each of the mlib_GraphicsFillCircle_X_*() functions draws a filled circle in Xor mode as follows:

\[
data[x, y] \ ^= c \ ^ \ c2
\]

Each of the mlib_GraphicsFillCircle_A_*() functions draws a filled circle with antialiasing.

Each of the mlib_GraphicsFillCircle_B_*() functions draws a filled circle with alpha blending as follows:

\[
data[x, y] = \frac{(data[x, y] \ \ * (255 - \ a) + c \ * \ a)}{255}
\]
Each of the `mlib_GraphicsFillCircle_AB_*()` functions draws a filled circle with antialiasing and alpha blending.

**Parameters**  Each of the functions takes some of the following arguments:
- `buffer` Pointer to the image into which the function is drawing.
- `x` X coordinate of the center.
- `y` Y coordinate of the center.
- `r` Radius of the arc.
- `c` Color used in the drawing.
- `c2` Alternation color.
- `a` Alpha value for blending. $0 \leq a \leq 255$.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_GraphicsFillArc(3MLIB), mlib_GraphicsFillEllipse(3MLIB), attributes(5)`
mlib_GraphicsFillEllipse, mlib_GraphicsFillEllipse_8, mlib_GraphicsFillEllipse_32,
mlib_GraphicsFillEllipse_X_8, mlib_GraphicsFillEllipse_X_32,
mlib_GraphicsFillEllipse_A_8, mlib_GraphicsFillEllipse_A_32,
mlib_GraphicsFillEllipse_B_8, mlib_GraphicsFillEllipse_B_32,
mlib_GraphicsFillEllipse_AB_8, mlib_GraphicsFillEllipse_AB_32 – draw filled ellipse

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_GraphicsFillEllipse_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsFillEllipse_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsFillEllipse_X_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFillEllipse_X_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFillEllipse_A_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsFillEllipse_A_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c);
mlib_status mlib_GraphicsFillEllipse_B_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
mlib_status mlib_GraphicsFillEllipse_B_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
mlib_status mlib_GraphicsFillEllipse_AB_8(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
mlib_status mlib_GraphicsFillEllipse_AB_32(mlib_image *buffer, mlib_s16 x,
mlib_s16 y, mlib_s32 a, mlib_s32 b, mlib_f32 t, mlib_s32 c, mlib_s32 alpha);
```

**Description**
Each of the mlib_GraphicsFillEllipse () functions draws a filled ellipse with the center at

\((x, y)\), major semiaxis \(a\), and minor semiaxis \(b\). The angle of the major semiaxis is \(t\)

clockwise from the X axis.

Each of the mlib_GraphicsFillEllipse_X () functions draws a filled ellipse in Xor mode

as follows:

\[
data[x, y] ^= c ^ c2
\]
Each of the `mlib_GraphicsFillEllipse_A_*( )` functions draws a filled ellipse with antialiasing.

Each of the `mlib_GraphicsFillEllipse_B_*( )` functions draws a filled ellipse with alpha blending as follows:
\[
data[x,y] = (data[x,y] \times (255 - \alpha) + c \times \alpha) / 255
\]

Each of the `mlib_GraphicsFillEllipse_A_*( )` functions draws a filled ellipse with antialiasing and alpha blending.

**Parameters** Each of the functions takes some of the following arguments:
- `buffer` Pointer to the image into which the function is drawing.
- `x` X coordinate of the center.
- `y` Y coordinate of the center.
- `a` Major semiaxis of the ellipse.
- `b` Minor semiaxis of the ellipse.
- `t` Angle of major semiaxis in radians.
- `c` Color used in the drawing.
- `c2` Alternation color.
- `alpha` Alpha value for blending. \[0 \leq \alpha \leq 255\].

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_GraphicsFillArc(3MLIB), mlib_GraphicsFillCircle(3MLIB), attributes(5)`

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFillPolygon_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillPolygon_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillPolygon_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsFillPolygon_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsFillPolygon_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillPolygon_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillPolygon_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
mlib_status mlib_GraphicsFillPolygon_G_8(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
    const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_G_32(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
    const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_Z_8(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    mlib_s32 c);

mlib_status mlib_GraphicsFillPolygon_Z_32(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    mlib_s32 c);

mlib_status mlib_GraphicsFillPolygon_AB_8(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
    mlib_s32 c, 
    mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_AB_32(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
    mlib_s32 c, 
    mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_ABG_8(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_ABG_32(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_ABGZ_8(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_ABGZ_32(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_ABZ_8(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    mlib_s32 c, 
    mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_ABZ_32(mlib_image *buffer, 
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
    const mlib_s16 *z, mlib_s32 npoints, 
    mlib_s32 c, 
    mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_AG_8(mlib_image *buffer, 
    const mlib_s16 *x, const mlib_s16 *y, 
    mlib_s32 npoints, 
    const mlib_s32 *c);
mlib_status mlib_GraphicsFillPolygon_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_BZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_BZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillPolygon_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_BZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_BZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillPolygon_GZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints);

mlib_status mlib_GraphicsFillPolygon_GZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);
const mlib_s16 *z, mlib_s32 *c);

description

Each of the mlib_GraphicsFillPolygon_*() functions draws a filled polygon enclosing (x1,y1), (x2,y2), ..., and (xn,yn).

Each of the mlib_GraphicsFillPolygon_X_*() functions draws a filled polygon in Xor mode as follows:

```
data[x,y] ^= c ^ c2
```

Each of the mlib_GraphicsFillPolygon_A_*() functions draws a filled polygon with antialiasing.

Each of the mlib_GraphicsFillPolygon_B_*() functions draws a filled polygon with alpha blending as follows:

```
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
```

Each of the mlib_GraphicsFillPolygon_G_*() functions draws a filled polygon with Gouraud shading.

Each of the mlib_GraphicsFillPolygon_Z_*() functions draws a filled polygon with Z buffering.

Each of the other functions draws a filled polygon with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

parameters

Each of the functions takes some of the following arguments:

- **buffer** Pointer to the image into which the function is drawing.
- **zbuffer** Pointer to the image that holds the Z buffer.
- **x** Pointer to the array of X coordinates of the vertices.
- **y** Pointer to the array of Y coordinates of the vertices.
- **z** Pointer to the array of Z coordinates of the vertices.
- **npoints** Number of vertices in the arrays.
- **c** Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- **c2** Alternation color.
- **a** Alpha value for blending. 0 \leq a \leq 255.

return values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_GraphicsFillPolygon(3MLIB), attributes(5)
REFERENCE

Multimedia Library Functions - Part 2

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFillRectangle_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c);
mlib_status mlib_GraphicsFillRectangle_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c);
mlib_status mlib_GraphicsFillRectangle_X_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFillRectangle_X_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFillRectangle_B_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsFillRectangle_B_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 w, mlib_s32 h, mlib_s32 c, mlib_s32 a);

Each of the mlib_GraphicsFillRectangle_*( ) functions draws a filled rectangle with the upper-left corner at (x, y), width w, and height h.

Each of the mlib_GraphicsFillRectangle_X_*( ) functions draws a filled rectangle in Xor mode as follows:

\[ \text{data}[x,y] \leftarrow \text{data}[x,y] \oplus c \oplus c_2 \]

Each of the mlib_GraphicsFillRectangle_B_*( ) functions draws a filled rectangle with alpha blending as follows:

\[ \text{data}[x,y] = \frac{(\text{data}[x,y] \times (255 - a) + c \times a)}{255} \]

Each of the functions takes some of the following arguments:

- buffer: Pointer to the image into which the function is drawing.
- x: X coordinate of the upper-left corner of the rectangle.
- y: Y coordinate of the upper-left corner of the rectangle.
- w: Width of the rectangle.
- h: Height of the rectangle.
- c: Color used in the drawing.
- c2: Alternation color.
Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_GraphicsDrawRectangle(3MLIB)`, attributes(5)
Name
mlib_GraphicsFillTriangle, mlib_GraphicsFillTriangle_8, mlib_GraphicsFillTriangle_32,
mlib_GraphicsFillTriangle_X_8, mlib_GraphicsFillTriangle_X_32,
mlib_GraphicsFillTriangle_A_8, mlib_GraphicsFillTriangle_A_32,
mlib_GraphicsFillTriangle_B_8, mlib_GraphicsFillTriangle_B_32,
mlib_GraphicsFillTriangle_G_8, mlib_GraphicsFillTriangle_G_32,
mlib_GraphicsFillTriangle_Z_8, mlib_GraphicsFillTriangle_Z_32,
mlib_GraphicsFillTriangle_AB_8, mlib_GraphicsFillTriangle_AB_32,
mlib_GraphicsFillTriangle_ABG_8, mlib_GraphicsFillTriangle_ABG_32,
mlib_GraphicsFillTriangle_ABGZ_8, mlib_GraphicsFillTriangle_ABGZ_32,
mlib_GraphicsFillTriangle_ABZ_8, mlib_GraphicsFillTriangle_ABZ_32,
mlib_GraphicsFillTriangle_AG_8, mlib_GraphicsFillTriangle_AG_32,
mlib_GraphicsFillTriangle_AGZ_8, mlib_GraphicsFillTriangle_AGZ_32,
mlib_GraphicsFillTriangle_AZ_8, mlib_GraphicsFillTriangle_AZ_32,
mlib_GraphicsFillTriangle_BG_8, mlib_GraphicsFillTriangle_BG_32,
mlib_GraphicsFillTriangle_BGZ_8, mlib_GraphicsFillTriangle_BGZ_32,
mlib_GraphicsFillTriangle_BZ_8, mlib_GraphicsFillTriangle_BZ_32,
mlib_GraphicsFillTriangle_GZ_8, mlib_GraphicsFillTriangle_GZ_32 – draw filled triangle

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFillTriangle_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_X_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsFillTriangle_X_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c, mlib_s32 c2);

mlib_status mlib_GraphicsFillTriangle_A_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_A_32(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3,
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_B_8(mlib_image *buffer, mlib_s16 x1,
mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3,
mlib_s16 y3, mlib_s32 c,
mlib_s32 w);
mlib_status mlib_GraphicsFillTriangle_B_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_G_8(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_G_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_Z_8(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_Z_32(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_AB_8(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_AB_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABG_8(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABG_32(mlib_image *buffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABGZ_8(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABGZ_32(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABZ_8(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_ABZ_32(mlib_image *buffer, mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, mlib_s32 c, mlib_s32 a);

mlib_GraphicsFillTriangle(3MLIB)
mlib_status mlib_GraphicsFillTriangle_ABZ_32(mlib_image *buffer, 
mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1, 
mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3, 
mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_AG_8(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_AGZ_32(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_AZ_8(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_AZ_32(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_BG_8(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_BG_32(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_BGZ_8(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_BGZ_32(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_BZ_8(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s32 c);

mlib_status mlib_GraphicsFillTriangle_BZ_32(mlib_image *buffer, 
mlib_s16 x1, mlib_s16 y1, mlib_s16 x2, mlib_s16 y2, mlib_s16 x3, mlib_s16 y3, 
mlib_s32 c);
mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_BZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangle_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2,
    mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

mlib_status mlib_GraphicsFillTriangle_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, mlib_s16 x1, mlib_s16 y1, mlib_s16 z1,
    mlib_s16 x2, mlib_s16 y2, mlib_s16 z2, mlib_s16 x3, mlib_s16 y3,
    mlib_s16 z3, mlib_s32 c1, mlib_s32 c2, mlib_s32 c3);

Each of the mlib_GraphicsFillTriangle_*() functions draws a filled triangle with the
vertices at (x1,y1), (x2,y2), and (x3,y3).

Each of the mlib_GraphicsFillTriangle_X_*() functions draws a filled triangle in Xor
mode as follows:

data[x,y] ^= c ^ c2

Each of the mlib_GraphicsFillTriangle_A_*() functions draws a filled triangle with
antialiasing.

Each of the mlib_GraphicsFillTriangle_B_*() functions draws a filled triangle with alpha
blending as follows:

data[x,y] = (data[x,y] * (255 - a) + c * a) / 255

Each of the mlib_GraphicsFillTriangle_G_*() functions draws a filled triangle with
Gouraud shading.

Each of the mlib_GraphicsFillTriangle_Z_*() functions draws a filled triangle with Z
buffering.

Each of the other functions draws a filled triangle with a combination of two or more features
like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

Parameters Each of the functions takes some of the following arguments:

buffer Pointer to the image into which the function is drawing.

zbuffer Pointer to the image that holds the Z buffer.

x1 X coordinate of the first vertex.

y1 Y coordinate of the first vertex.
z1 Z coordinate of the first vertex.
x2 X coordinate of the second vertex.
y2 Y coordinate of the second vertex.
z2 Z coordinate of the second vertex.
x3 X coordinate of the third vertex.
y3 Y coordinate of the third vertex.
z3 Z coordinate of the third vertex.
c Color used in the drawing.
c1 Color of the first vertex.
c2 Color of the second vertex, or the alternation color in Xor Mode.
c3 Color of the third vertex.
a Alpha value for blending. 0 ≤ a ≤ 255.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_GraphicsFillTriangle(3MLIB), attributes(5)
Name

mlib_GraphicsFillTriangleFanSet, mlib_GraphicsFillTriangleFanSet_8,
mlib_GraphicsFillTriangleFanSet_32, mlib_GraphicsFillTriangleFanSet_X_8,
mlib_GraphicsFillTriangleFanSet_X_32, mlib_GraphicsFillTriangleFanSet_A_8,
mlib_GraphicsFillTriangleFanSet_A_32, mlib_GraphicsFillTriangleFanSet_B_8,
mlib_GraphicsFillTriangleFanSet_B_32, mlib_GraphicsFillTriangleFanSet_G_8,
mlib_GraphicsFillTriangleFanSet_G_32, mlib_GraphicsFillTriangleFanSet_Z_8,
mlib_GraphicsFillTriangleFanSet_Z_32, mlib_GraphicsFillTriangleFanSet_AB_8,
mlib_GraphicsFillTriangleFanSet_AB_32, mlib_GraphicsFillTriangleFanSet_ABG_8,
mlib_GraphicsFillTriangleFanSet_ABG_32, mlib_GraphicsFillTriangleFanSet_ABGZ_8,
mlib_GraphicsFillTriangleFanSet_ABGZ_32, mlib_GraphicsFillTriangleFanSet_ABZ_8,
mlib_GraphicsFillTriangleFanSet_ABZ_32, mlib_GraphicsFillTriangleFanSet_AG_8,
mlib_GraphicsFillTriangleFanSet_AG_32, mlib_GraphicsFillTriangleFanSet_AGZ_8,
mlib_GraphicsFillTriangleFanSet_AGZ_32, mlib_GraphicsFillTriangleFanSet_AZ_8,
mlib_GraphicsFillTriangleFanSet_AZ_32, mlib_GraphicsFillTriangleFanSet_BG_8,
mlib_GraphicsFillTriangleFanSet_BG_32, mlib_GraphicsFillTriangleFanSet_BGZ_8,
mlib_GraphicsFillTriangleFanSet_BGZ_32, mlib_GraphicsFillTriangleFanSet_BZ_8,
mlib_GraphicsFillTriangleFanSet_BZ_32, mlib_GraphicsFillTriangleFanSet_GZ_8,
mlib_GraphicsFillTriangleFanSet_GZ_32 – draw filled triangle set where all members of the
set have a common vertex

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFillTriangleFanSet_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_X_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_X_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_A_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_A_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_B_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_B_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_G_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_G_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_Z_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_Z_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AB_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AB_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_ABG_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_ABG_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_ABGZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_ABGZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_ABZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_ABZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AG_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AG_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AGZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AGZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_AZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BG_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BG_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BGZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BGZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_GZ_8(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_GZ_32(mlib_image *buffer,
   const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
   mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_G_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleFanSet_G_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleFanSet_Z_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleFanSet_Z_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleFanSet_AB_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_AB_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_ABG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_ABG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_ABGZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_ABGZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_ABZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_ABZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleFanSet_AG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleFanSet_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);
mlib_status mlib_GraphicsFillTriangleFanSet_AGZ_8(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
mlib_status mlib_GraphicsFillTriangleFanSet_AGZ_32(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
mlib_status mlib_GraphicsFillTriangleFanSet_AZ_8(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_AZ_32(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleFanSet_BG_8(mlib_image *buffer, 
  const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
  const mlib_s32 *c, mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BG_32(mlib_image *buffer, 
  const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, 
  const mlib_s32 *c, mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BGZ_8(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BGZ_32(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BZ_8(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_BZ_32(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleFanSet_GZ_8(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
mlib_status mlib_GraphicsFillTriangleFanSet_GZ_32(mlib_image *buffer, 
  mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, 
  const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

Description Each of the mlib_GraphicsFillTriangleFanSet_*( ) functions draws a set of filled triangles 
with vertices at \{(x_1,y_1), (x_2,y_2), (x_3,y_3), \ldots, (x_n,y_n)\}.

Each of the mlib_GraphicsFillTriangleFanSet_X_*( ) functions draws a set of filled 
triangles in Xor mode as follows:
Each of the `mlib_GraphicsFillTriangleFanSet_A_*()` functions draws a set of filled triangles with antialiasing.

Each of the `mlib_GraphicsFillTriangleFanSet_B_*()` functions draws a set of filled triangles with alpha blending as follows:

\[
data[x,y] = (data[x,y] \times (255 - a) + c \times a) / 255
\]

Each of the `mlib_GraphicsFillTriangleFanSet_G_*()` functions draws a set of filled triangles with Gouraud shading.

Each of the `mlib_GraphicsFillTriangleFanSet_Z_*()` functions draws a set of filled triangles with Z buffering.

Each of the other functions draws a set of filled triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

**Parameters**

Each of the functions takes some of the following arguments:

- `buffer` Pointer to the image into which the function is drawing.
- `zbuffer` Pointer to the image that holds the Z buffer.
- `x` Pointer to array of X coordinates of the points.
- `y` Pointer to array of Y coordinates of the points.
- `z` Pointer to array of Z coordinates of the points.
- `npoints` Number of points in the arrays.
- `c` Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- `c2` Alternation color.
- `a` Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

mlib_GraphicsFillTriangle(3MLIB), mlib_GraphicsFillTriangleSet(3MLIB),
mlib_GraphicsFillTriangleStripSet(3MLIB), attributes(5)
mlib_GraphicsFillTriangleSet(3MLIB)

Name
mlib_GraphicsFillTriangleSet, mlib_GraphicsFillTriangleSet_8,
mlib_GraphicsFillTriangleSet_32, mlib_GraphicsFillTriangleSet_X_8,
mlib_GraphicsFillTriangleSet_X_32, mlib_GraphicsFillTriangleSet_A_8,
mlib_GraphicsFillTriangleSet_A_32, mlib_GraphicsFillTriangleSet_B_8,
mlib_GraphicsFillTriangleSet_B_32, mlib_GraphicsFillTriangleSet_G_8,
mlib_GraphicsFillTriangleSet_G_32, mlib_GraphicsFillTriangleSet_Z_8,
mlib_GraphicsFillTriangleSet_Z_32, mlib_GraphicsFillTriangleSet_AB_8,
mlib_GraphicsFillTriangleSet_AB_32, mlib_GraphicsFillTriangleSet_ABG_8,
mlib_GraphicsFillTriangleSet_ABG_32, mlib_GraphicsFillTriangleSet_ABGZ_8,
mlib_GraphicsFillTriangleSet_ABGZ_32, mlib_GraphicsFillTriangleSet_ABZ_8,
mlib_GraphicsFillTriangleSet_ABZ_32, mlib_GraphicsFillTriangleSet_AG_8,
mlib_GraphicsFillTriangleSet_AG_32, mlib_GraphicsFillTriangleSet_AGZ_8,
mlib_GraphicsFillTriangleSet_AGZ_32, mlib_GraphicsFillTriangleSet_AZ_8,
mlib_GraphicsFillTriangleSet_AZ_32, mlib_GraphicsFillTriangleSet_BG_8,
mlib_GraphicsFillTriangleSet_BG_32, mlib_GraphicsFillTriangleSet_BGZ_8,
mlib_GraphicsFillTriangleSet_BGZ_32, mlib_GraphicsFillTriangleSet_BGZ_8,
mlib_GraphicsFillTriangleSet_BGZ_32, mlib_GraphicsFillTriangleSet_GZ_8,
mlib_GraphicsFillTriangleSet_GZ_32 – draw filled triangle set where each member can have
different vertices

Synopsis
cc [ flag... ] file... -tlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFillTriangleSet_8(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c);
mlib_status mlib_GraphicsFillTriangleSet_32(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c);
mlib_status mlib_GraphicsFillTriangleSet_X_8(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c,
    mib_s32 c2);
mlib_status mlib_GraphicsFillTriangleSet_X_32(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c,
    mib_s32 c2);
mlib_status mlib_GraphicsFillTriangleSet_A_8(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c);
mlib_status mlib_GraphicsFillTriangleSet_A_32(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c);
mlib_status mlib_GraphicsFillTriangleSet_B_8(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c,
    mib_s32 a);
mlib_status mlib_GraphicsFillTriangleSet_B_32(mlib_image *buffer,
    const mib_s16 *x, const mib_s16 *y, mib_s32 npoints, mib_s32 c,
    mib_s32 a);
mlib_status mlib_GraphicsFillTriangleSet_G_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleSet_G_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleSet_Z_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleSet_Z_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleSet_AB_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_AB_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_ABG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_ABG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_ABGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_ABGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_ABZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_ABZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleSet_AG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleSet_AG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);
Each of the `mlib_GraphicsFillTriangleSet()` functions draws a set of filled triangles with vertices at \((x_1,y_1), (x_2,y_2), (x_3,y_3)\), \((x_4,y_4), (x_5,y_5), (x_6,y_6)\), ..., and \((x_{n-2},y_{n-2}), (x_{n-1},y_{n-1}), (x_n,y_n)\).

Each of the `mlib_GraphicsFillTriangleSet_X_()` functions draws a set of filled triangles in Xor mode as follows:
Each of the \texttt{mlib\_GraphicsFillTriangleSet\_A\_\(*\)} functions draws a set of filled triangles with antialiasing.

Each of the \texttt{mlib\_GraphicsFillTriangleSet\_B\_\(*\)} functions draws a set of filled triangles with alpha blending as follows:

\[
data[x, y] = (\text{data}[x, y] \times (255 - a) + c \times a) / 255
\]

Each of the \texttt{mlib\_GraphicsFillTriangleSet\_G\_\(*\)} functions draws a set of filled triangles with Gouraud shading.

Each of the \texttt{mlib\_GraphicsFillTriangleSet\_Z\_\(*\)} functions draws a set of filled triangles with Z buffering.

Each of the other functions draws a set of filled triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

**Parameters**

Each of the functions takes some of the following arguments:

- \textit{buffer} Pointer to the image into which the function is drawing.
- \textit{zbuffer} Pointer to the image that holds the Z buffer.
- \textit{x} Pointer to array of X coordinates of the points.
- \textit{y} Pointer to array of Y coordinates of the points.
- \textit{z} Pointer to array of Z coordinates of the points.
- \textit{npoints} Number of points in the arrays. \textit{npoints} must be a multiple of 3.
- \textit{c} Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- \textit{c2} Alternation color.
- \textit{a} Alpha value for blending. \(0 \leq a \leq 255\).

**Return Values**

Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

**Attributes**

See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_GraphicsFillTriangle(3MLIB), mlib_GraphicsFillTriangleFanSet(3MLIB),
mlib_GraphicsFillTriangleStripSet(3MLIB), attributes(5)
mlib_GraphicsFillTriangleStripSet, mlib_GraphicsFillTriangleStripSet_8, 
mlib_GraphicsFillTriangleStripSet_32, mlib_GraphicsFillTriangleStripSet_X_8, 
mlib_GraphicsFillTriangleStripSet_X_32, mlib_GraphicsFillTriangleStripSet_A_8, 
mlib_GraphicsFillTriangleStripSet_A_32, mlib_GraphicsFillTriangleStripSet_B_8, 
mlib_GraphicsFillTriangleStripSet_B_32, mlib_GraphicsFillTriangleStripSet_G_8, 
mlib_GraphicsFillTriangleStripSet_G_32, mlib_GraphicsFillTriangleStripSet_Z_8, 
mlib_GraphicsFillTriangleStripSet_Z_32, mlib_GraphicsFillTriangleStripSet_AB_8, 
mlib_GraphicsFillTriangleStripSet_AB_32, mlib_GraphicsFillTriangleStripSet_ABG_8, 
mlib_GraphicsFillTriangleStripSet_ABG_32, mlib_GraphicsFillTriangleStripSet_ABGZ_8, 
mlib_GraphicsFillTriangleStripSet_ABGZ_32, mlib_GraphicsFillTriangleStripSet_ABZ_8, 
mlib_GraphicsFillTriangleStripSet_ABZ_32, mlib_GraphicsFillTriangleStripSet_AG_8, 
mlib_GraphicsFillTriangleStripSet_AG_32, mlib_GraphicsFillTriangleStripSet_AGZ_8, 
mlib_GraphicsFillTriangleStripSet_AGZ_32, mlib_GraphicsFillTriangleStripSet_AZ_8, 
mlib_GraphicsFillTriangleStripSet_AZ_32, mlib_GraphicsFillTriangleStripSet_BG_8, 
mlib_GraphicsFillTriangleStripSet_BG_32, mlib_GraphicsFillTriangleStripSet_BGZ_8, 
mlib_GraphicsFillTriangleStripSet_BGZ_32, mlib_GraphicsFillTriangleStripSet_BZ_8, 
mlib_GraphicsFillTriangleStripSet_BZ_32, mlib_GraphicsFillTriangleStripSet_GZ_8, 
mlib_GraphicsFillTriangleStripSet_GZ_32 – draw filled triangle set where the first side of 
each member is common to the second side of the previous member

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFillTriangleStripSet_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleStripSet_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleStripSet_X_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleStripSet_X_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleStripSet_A_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleStripSet_A_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c);
mlib_status mlib_GraphicsFillTriangleStripSet_B_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleStripSet_B_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c,
    mlib_s32 a);
mlib_status mlib_GraphicsFillTriangleStripSet_G_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleStripSet_G_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleStripSet_Z_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleStripSet_Z_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleStripSet_AB_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_AB_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_ABG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_ABG_32(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_ABGZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_ABGZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_ABZ_8(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_ABZ_32(mlib_image *buffer, mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_AG_8(mlib_image *buffer, const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints, const mlib_s32 *c);

mlib_GraphicsFillTriangleStripSet(3MLIB)
mlib_status mlib_GraphicsFillTriangleStripSet_AG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleStripSet_AGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleStripSet_AGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleStripSet_AZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleStripSet_AZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, mlib_s32 c);

mlib_status mlib_GraphicsFillTriangleStripSet_BG_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_BG_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_BGZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_BGZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_BZ_8(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_BZ_32(mlib_image *buffer,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 npoints,
    mlib_s32 c, mlib_s32 a);

mlib_status mlib_GraphicsFillTriangleStripSet_GZ_8(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);

mlib_status mlib_GraphicsFillTriangleStripSet_GZ_32(mlib_image *buffer,
    mlib_image *zbuffer, const mlib_s16 *x, const mlib_s16 *y,
    const mlib_s16 *z, mlib_s32 npoints, const mlib_s32 *c);
Each of the `mlib_GraphicsFillTriangleStripSet_*()` functions draws a set of filled triangles with vertices at \{(x_1,y_1), (x_2,y_2), (x_3,y_3)\}, \{(x_2,y_2), (x_3,y_3), (x_4,y_4)\}, ..., and \{(x_{n-2},y_{n-2}), (x_{n-1},y_{n-1}), (x_n,y_n)\}.

Each of the `mlib_GraphicsFillTriangleStripSet_X_*()` functions draws a set of filled triangles in Xor mode as follows:

\[
data[x,y] ^= c ^ c_2
\]

Each of the `mlib_GraphicsFillTriangleStripSet_A_*()` functions draws a set of filled triangles with antialiasing.

Each of the `mlib_GraphicsFillTriangleStripSet_B_*()` functions draws a set of filled triangles with alpha blending as follows:

\[
data[x,y] = (data[x,y] * (255 - a) + c * a) / 255
\]

Each of the `mlib_GraphicsFillTriangleStripSet_G_*()` functions draws a set of filled triangles with Gouraud shading.

Each of the `mlib_GraphicsFillTriangleStripSet_Z_*()` functions draws a set of filled triangles with Z buffering.

Each of the other functions draws a set of filled triangles with a combination of two or more features like antialiasing (A), alpha blending (B), Gouraud shading (G), and Z buffering (Z).

### Parameters
Each of the functions takes some of the following arguments:

- **buffer** Pointer to the image into which the function is drawing.
- **zbuffer** Pointer to the image that holds the Z buffer.
- **x** Pointer to array of X coordinates of the points.
- **y** Pointer to array of Y coordinates of the points.
- **z** Pointer to array of Z coordinates of the points.
- **npoints** Number of points in the arrays.
- **c** Color used in the drawing, or pointer to array of colors of the points in the case of Gouraud shading.
- **c2** Alternation color.
- **a** Alpha value for blending. \(0 \leq a \leq 255\).

### Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:
### mlib_GraphicsFillTriangleStripSet(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_GraphicsFillTriangle(3MLIB), mlib_GraphicsFillTriangleSet(3MLIB), mlib_GraphicsFillTriangleFanSet(3MLIB), attributes(5)
mlib_GraphicsFloodFill_8, mlib_GraphicsFloodFill_32 – flood fill

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_GraphicsFloodFill_8(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 c, mlib_s32 c2);
mlib_status mlib_GraphicsFloodFill_32(mlib_image *buffer, mlib_s16 x, mlib_s16 y, mlib_s32 c, mlib_s32 c2);
```

Description

Each of these functions performs flood fill.

Parameters

Each of the functions takes the following arguments:

- **buffer**  Pointer to the image into which the function is drawing.
- **x**  X coordinate of the starting point.
- **y**  Y coordinate of the starting point.
- **c**  Color used in the drawing.
- **c2**  Color that defines the filling interior.

Return Values

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`attributes(5)`
mlib_ImageAbs(3MLIB)

Name
mlib_ImageAbs – computes the absolute value of the image pixels

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAbs(mlib_image *dst, const mlib_image *src);

Description
The mlib_ImageAbs() function computes the absolute value of the image pixels.

It uses the following equation:

\[ dst[x][y][i] = |src[x][y][i]| \]

Parameters
The function takes the following arguments:

- **dst**  Pointer to destination image.
- **src**  Pointer to source image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageAbs_Fp(3MLIB), mlib_ImageAbs_Fp_Inp(3MLIB), mlib_ImageAbs_Inp(3MLIB), attributes(5)
mlib_ImageAbs_Fp(3MLIB)

Name       mlib_ImageAbs_Fp - computes the absolute value of the image pixels

Synopsis   cc [ flag... ] file... -lmlib [ library... ]
            #include <mlib.h>
            
            mlib_status mlib_ImageAbs_Fp(mlib_image *dst, const mlib_image *src);

Description The mlib_ImageAbs_Fp() function computes the floating-point absolute value of the image pixels.

It uses the following equation:

dst[x][y][i] = |src[x][y][i]|

Parameters The function takes the following arguments:

    dst        Pointer to destination image.
    src        Pointer to source image.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageAbs(3MLIB), mlib_ImageAbs_Fp_Inp(3MLIB), mlib_ImageAbs_Inp(3MLIB), attributes(5)
mlib_ImageAbs_Fp_Inp() function computes the floating-point absolute value of the image pixels, in place.

It uses the following equation:

\[ srcdst[x][y][i] = |srcdst[x][y][i]| \]

**Parameters**
The function takes the following arguments:

- `srcdst` Pointer to source and destination image.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageAbs(3MLIB), mlib_ImageAbs_Fp(3MLIB), mlib_ImageAbs_Inp(3MLIB), attributes(5)
The `mlib_ImageAbs_Inp()` function computes the absolute value of the image pixels in place.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = |\text{srcdst}[x][y][i]| \]

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageAbs(3MLIB), mlib_ImageAbs_Fp(3MLIB), mlib_ImageAbs_Fp_Inp(3MLIB)`
Name  
mlib_ImageAdd – computes the addition of two images on a pixel-by-pixel basis

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAdd(mlib_image dst, const mlib_image *src1,
const mlib_image *src2);

Description  
The mlib_ImageAdd() function computes the addition of two images on a pixel-by-pixel basis.

It uses the following equation:

dst[x][y][i] = src1[x][y][i] + src2[x][y][i]

Parameters  
The function takes the following arguments:

dst  Pointer to destination image.
src1  Pointer to first source image.
src2  Pointer to second source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageAdd_Fp(3MLIB), mlib_ImageAdd_Fp_Inp(3MLIB), mlib_ImageAdd_Inp(3MLIB), attributes(5)
mlib_ImageAdd_Fp() computes the addition of two images on a pixel-by-pixel basis.

### Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAdd_Fp(mlib_image *dst,
              const mlib_image *src1,
              const mlib_image *src2);
```

### Description

The `mlib_ImageAdd_Fp()` function computes the addition of two floating-point images on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = \text{src1}[x][y][i] + \text{src2}[x][y][i]
\]

### Parameters

- **dst**: Pointer to destination image.
- **src1**: Pointer to first source image.
- **src2**: Pointer to second source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise, it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageAdd(3MLIB), mlib_ImageAdd_Fp_Inp(3MLIB), mlib_ImageAdd_Inp(3MLIB), attributes(5)`
The `mlib_ImageAdd_Fp_Inp` function computes the addition of two floating-point images on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src1dst[x][y][i] = src1dst[x][y][i] + src2[x][y][i]
```

The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageAdd(3MLIB), mlib_ImageAdd_Fp(3MLIB), mlib_ImageAdd_Inp(3MLIB), attributes(5)`
Name  mlib_ImageAdd_Inp – computes the addition of two images on a pixel-by-pixel basis, in place

Synopsis  cc [ flag ] file ... -lmlib [ library ... ]
           #include <mlib.h>

           mlib_status mlib_ImageAdd_Inp(mlib_image *src1dst, const mlib_image *src2);

Description  The mlib_ImageAdd_Inp() function computes the addition of two images on a pixel-by-pixel basis, in place.

It uses the following equation:

src1dst[x][y][i] = src1dst[x][y][i] + src2[x][y][i]

Parameters  The function takes the following arguments:

src1dst  Pointer to first source and destination image.

src2  Pointer to second source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageAdd(3MLIB), mlib_ImageAdd_Fp(3MLIB), mlib_ImageAdd_Fp_Inp(3MLIB), attributes(5)
mlib_ImageAffine function does affine transformation on an image according to the following equation:

\[
\begin{align*}
xd &= a \times xs + b \times ys + tx \\
yd &= c \times xs + d \times ys + ty
\end{align*}
\]

where a point with coordinates (xs, ys) in the source image is mapped to a point with coordinates (xd, yd) in the destination image.

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

### Parameters
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **filter** Type of resampling filter. It can be one of the following:
  - MLIB_NEAREST
  - MLIB_BILINEAR
  - MLIB_BICUBIC
  - MLIB_BICUBIC2
- **edge** Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_OP_NEAREST
  - MLIB_EDGE_SRC_EXTEND
  - MLIB_EDGE_SRC_PADDED
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageAffine_Fp(3MLIB), mlib_ImageAffineIndex(3MLIB),
mlib_ImageAffineTransform(3MLIB), mlib_ImageAffineTransform_Fp(3MLIB),
mlib_ImageAffineTransformIndex(3MLIB), mlib_ImageSetPaddings(3MLIB), attributes(5)
The `mlib_ImageAffine_Fp()` function does affine transformation on a floating-point image according to the following equation:

\[
\begin{align*}
xd &= a \cdot xs + b \cdot ys + tx \\
zd &= c \cdot xs + d \cdot ys + ty
\end{align*}
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, zd)\) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **mtx**: Transformation matrix. \(m[0]\) holds \(a\); \(m[1]\) holds \(b\); \(m[2]\) holds \(tx\); \(m[3]\) holds \(c\); \(m[4]\) holds \(d\); \(m[5]\) holds \(ty\).
- **filter**: Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`. 

```c
#include <mlib.h>

mlib_status mlib_ImageAffine_Fp(mlib_image *dst, const mlib_image *src,
                                const mlib_d64 *mtx, mlib_filter filter, mlib_edge edge);
```
Attributes  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  `mlib_ImageAffine(3MLIB), mlib_ImageAffineIndex(3MLIB), mlib_ImageAffineTransform(3MLIB), mlib_ImageAffineTransform_Fp(3MLIB), mlib_ImageAffineTransformIndex(3MLIB), mlib_ImageSetPaddings(3MLIB), attributes(5)`
mlib_ImageAffineIndex function does affine transformation on a color indexed image according to the following equation:

\[
    \begin{align*}
    x_d &= a x_s + b y_s + t_x \\
    y_d &= c x_s + d y_s + t_y
    \end{align*}
\]

where a point with coordinates \((x_s, y_s)\) in the source image is mapped to a point with coordinates \((x_d, y_d)\) in the destination image.

The image data type must be \texttt{MLIB\_BYTE} or \texttt{MLIB\_SHORT}.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

The function takes the following arguments:

- \texttt{dst} Pointer to destination image.
- \texttt{src} Pointer to source image.
- \texttt{mtx} Transformation matrix. \texttt{mtx[0]} holds \(a\); \texttt{mtx[1]} holds \(b\); \texttt{mtx[2]} holds \(t_x\); \texttt{mtx[3]} holds \(c\); \texttt{mtx[4]} holds \(d\); \texttt{mtx[5]} holds \(t_y\).
- \texttt{filter} Type of resampling filter. It can be one of the following:
  - \texttt{MLIB\_NEAREST}
  - \texttt{MLIB\_BILINEAR}
  - \texttt{MLIB\_BICUBIC}
  - \texttt{MLIB\_BICUBIC2}
- \texttt{edge} Type of edge condition. It can be one of the following:
  - \texttt{MLIB\_EDGE\_DST\_NO\_WRITE}
  - \texttt{MLIB\_EDGE\_DST\_FILL\_ZERO}
  - \texttt{MLIB\_EDGE\_OP\_NEAREST}
  - \texttt{MLIB\_EDGE\_SRC\_EXTEND}
  - \texttt{MLIB\_EDGE\_SRC\_Padded}

\texttt{colormap} Internal data structure for inverse color mapping. This data structure is generated by the \texttt{mlib\_ImageColorTrue2IndexInit()} function.
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageAffine(3MLIB), mlib_ImageAffine_Fp(3MLIB),
          mlib_ImageAffineTransform(3MLIB), mlib_ImageAffineTransform_Fp(3MLIB),
          mlib_ImageAffineTransformIndex(3MLIB), attributes(5)
The `mlib_ImageAffineTable()` function does affine transformation on an image with table-driven interpolation.

The following equation represents the affine transformation:

\[
\begin{align*}
xd &= a \cdot xs + b \cdot ys + tx \\
yd &= c \cdot xs + d \cdot ys + ty 
\end{align*}
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

### Parameters

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **mtx** Transformation matrix. \(mtx[0]\) holds \(a\); \(mtx[1]\) holds \(b\); \(mtx[2]\) holds \(tx\); \(mtx[3]\) holds \(c\); \(mtx[4]\) holds \(d\); \(mtx[5]\) holds \(ty\).
- **interp_table** Pointer to an interpolation table. The table is created by the `mlib_ImageInterpTableCreate()` function.
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:
### mlib_ImageAffineTable(3MLIB)

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also:  
- `mlib_ImageInterpTableCreate(3MLIB)`  
- `mlib_ImageInterpTableDelete(3MLIB)`  
- `mlib_ImageAffineTable_Fp(3MLIB)`  
- `mlib_ImageAffine(3MLIB)`  
- `mlib_ImageAffine_Fp(3MLIB)`  
- `attributes(5)`
mlib_ImageAffineTable_Fp – affine transformation on an image with table-driven interpolation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_ImageAffineTable_Fp(mlib_image *dst,
    const mlib_image *src, const mlib_d64 *mtx,
    const void *interp_table, mlib_edge edge);

Description

The `mlib_ImageAffineTable_Fp()` function does affine transformation on a floating-point image with table-driven interpolation.

The following equation represents the affine transformation:

\[
\begin{align*}
xd &= a \times xs + b \times ys + tx \\
yd &= c \times xs + d \times ys + ty
\end{align*}
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

Parameters

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **mtx** Transformation matrix. \(mtx[0]\) holds \(a\); \(mtx[1]\) holds \(b\); \(mtx[2]\) holds \(tx\);
  \(mtx[3]\) holds \(c\); \(mtx[4]\) holds \(d\); \(mtx[5]\) holds \(ty\).
- **interp_table** Pointer to an interpolation table. The table is created by the `mlib_ImageInterpTableCreate()` function.
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`. 
mlib_ImageAffineTable_Fp(3MLIB)

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageInterpTableCreate(3MLIB), mlib_ImageInterpTableDelete(3MLIB),  
mlib_ImageAffineTable(3MLIB), mlib_ImageAffine(3MLIB),  
mlib_ImageAffine_Fp(3MLIB), attributes(5)
The `mlib_ImageAffineTransform()` function does affine transformation on an image, checking the matrix first and taking advantage of special cases. The following equation represents the affine transformation:

\[
xd = ax + by + tx \\
yd = cx + dy + ty
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageAffine(3MLIB), mlib_ImageAffine_Fp(3MLIB),  
mlib_ImageAffineIndex(3MLIB), mlib_ImageAffineTransform_Fp(3MLIB),  
mlib_ImageAffineTransformIndex(3MLIB), mlib_ImageSetPaddings(3MLIB),  
attributes(5)
The `mlib_ImageAffineTransform_Fp()` function does affine transformation on a floating-point image, checking the matrix first and taking advantage of special cases.

The following equation represents the affine transformation:

\[
\begin{align*}
xd &= a \times xs + b \times ys + tx \\
yd &= c \times xs + d \times ys + ty
\end{align*}
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

**Parameters**

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **mtx**: Transformation matrix. \(mtx[0]\) holds \(a\); \(mtx[1]\) holds \(b\); \(mtx[2]\) holds \(tx\); \(mtx[3]\) holds \(c\); \(mtx[4]\) holds \(d\); \(mtx[5]\) holds \(ty\).
- **filter**: Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_ImageAffine(3MLIB)`, `mlib_ImageAffine_Fp(3MLIB)`, `mlib_ImageAffineIndex(3MLIB)`, `mlib_ImageAffineTransform(3MLIB)`, `mlib_ImageAffineTransformIndex(3MLIB)`, `mlib_ImageSetPaddings(3MLIB)`, attributes(5)
mlib_ImageAffineTransformIndex(3MLIB)

Name  mlib_ImageAffineTransformIndex – affine transformation on a color indexed image, checking the matrix first

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAffineTransformIndex(mlib_image *dst, const mlib_image *src, const mlib_d64 *mtx, mlib_filter filter, mlib_edge edge, const void *colormap);

Description  The mlib_ImageAffineTransformIndex() function does affine transformation on a color indexed image, checking the matrix first and taking advantage of special cases.

The following equation represents the affine transformation:

\[
\begin{align*}
xd &= a \cdot xs + b \cdot ys + tx \\
yd &= c \cdot xs + d \cdot ys + ty
\end{align*}
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The image data type must be MLIB_BYTE or MLIB_SHORT.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

Parameters  The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **mtx**: Transformation matrix. \(\text{mtx}[0]\) holds \(a\); \(\text{mtx}[1]\) holds \(b\); \(\text{mtx}[2]\) holds \(tx\); \(\text{mtx}[3]\) holds \(c\); \(\text{mtx}[4]\) holds \(d\); \(\text{mtx}[5]\) holds \(ty\).
- **filter**: Type of resampling filter. It can be one of the following:
  - MLIB_NEAREST
  - MLIB_BILINEAR
  - MLIB_BICUBIC
  - MLIB_BICUBIC2
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_OP_NEAREST
  - MLIB_EDGE_SRC_EXTEND
  - MLIB_EDGE_SRC_PADDED
**colormap**  Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_ImageAffine(3MLIB)`, `mlib_ImageAffine_Fp(3MLIB)`, `mlib_ImageAffineIndex(3MLIB)`, `mlib_ImageAffineTransform(3MLIB)`, `mlib_ImageAffineTransform_Fp(3MLIB)`, `attributes(5)`
mlib_ImageAnd(3MLIB)

Name  
mlib_ImageAnd – computes the And of two images

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAnd(mlib_image *dst, const mlib_image *src1,
const mlib_image *src2);

Description  
The mlib_ImageAnd() function computes the And of two images according to the following equation:

dst[x][y][i] = src1[x][y][i] & src2[x][y][i]

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

Parameters  
The function takes the following arguments:

dst Pointer to destination image.
src1 Pointer to first source image.
src2 Pointer to second source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageAnd_Inp(3MLIB), attributes(5)
Name  
mlib_ImageAnd_Inp – computes the And of two image, in place

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAnd_Inp(mlib_image *src1dst, const mlib_image *src2);

Description  
The mlib_ImageAnd_Inp() function computes the And of two images, in place, according to the following equation:

csrc1dst[x][y][i] = src1dst[x][y][i] & src2[x][y][i]

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

Parameters  
The function takes the following arguments:

csrc1dst  Pointer to first source and destination image.
csrc2  Pointer to second source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageAnd(3MLIB), attributes(5)
mlib_ImageAndNot1_Inp(3MLIB)

Name

mlib_ImageAndNot1_Inp – computes the And of the first source image and the Not of the second source image, in place

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAndNot1_Inp(mlib_image *src1dst,
const mlib_image *src2);

Description

The mlib_ImageAndNot1_Inp() function computes the logical Not of the second source image and then computes the logical And of that result with the first source image, on a pixel-by-pixel basis, and stores the final result in the first source image. It uses the following equation:

src1dst[x][y][i] = src1dst[x][y][i] & (~src2[x][y][i])

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

Parameters

The function takes the following arguments:

src1dst Pointer to first source and destination image.
src2 Pointer to second source image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageAndNot(3MLIB), mlib_ImageAndNot2_Inp(3MLIB), attributes(5)
### mlib_ImageAndNot2_Inp(3MLIB)

**Name**  
mlib_ImageAndNot2_Inp – computes the And of the first source image and the Not of the second source image, in place

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_ImageAndNot2_Inp(mlib_image *src2dst, const mlib_image *src1);
```

**Description**  
The `mlib_ImageAndNot2_Inp()` function computes the logical Not of the second source image and then computes the logical And of that result with the first source image, on a pixel-by-pixel basis, and stores the final result in the second source image. It uses the following equation:

\[
src2dst[x][y][i] = src1[x][y][i] \& (~src2dst[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**  
The function takes the following arguments:

- `src2dst`  
  Pointer to second source and destination image.

- `src1`  
  Pointer to first source image.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_ImageAnd(3MLIB), mlib_ImageAnd_Inp(3MLIB), attributes(5)`
The `mlib_ImageAndNot()` function computes the logical Not of the second source image and then computes the logical And of the result with the first source image, on a pixel-by-pixel basis. It uses the following equation:

\[
dst[x][y][i] = src1[x][y][i] \& \neg src2[x][y][i]
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageAndNot1_Inp(3MLIB)`, `mlib_ImageAndNot2_Inp(3MLIB)`, attributes(5)
# include <mlib.h>

mlib_status mlib_ImageAutoCorrel(mlib_d64 *correl, const mlib_image *img,
                               mlib_s32 dx, mlib_s32 dy);

The `mlib_ImageAutoCorrel()` function computes the auto-correlation of an image, given an offset.

It uses the following equation:

\[
\text{correl}[i] = \frac{1}{(w-dx) \cdot (h-dy)} \sum_{x=0}^{w-dx-1} \sum_{y=0}^{h-dy-1} \text{img}[x][y][i] \cdot \text{img}[x+dx][y+dy][i]
\]

where \( w \) and \( h \) are the width and height of the image, respectively.

## Parameters
The function takes the following arguments:

- `correl`  
  Pointer to auto-correlation array where size is equal to the number of channels. `correl[i]` contains the auto-correlation of channel \( i \).

- `img`  
  Pointer to image.

- `dx`  
  Displacement in the X direction.

- `dy`  
  Displacement in the Y direction.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageAutoCorrel_Fp(3MLIB)`, `mlib_ImageCrossCorrel(3MLIB)`,
- `mlib_ImageCrossCorrel_Fp(3MLIB)`, attributes(5)
# mlib_ImageAutoCorrel_Fp

The `mlib_ImageAutoCorrel_Fp()` function computes the auto-correlation of a floating-point image, given an offset.

It uses the following equation:

\[
\text{correl}[i] = \frac{1}{(w-dx)(h-dy)} \sum_{x=0}^{w-dx-1} \sum_{y=0}^{h-dy-1} \text{img}[x][y][i] \times \text{img}[x+dx][y+dy][i]
\]

where \( w \) and \( h \) are the width and height of the image, respectively.

## Parameters

- **correl**: Pointer to auto-correlation array where size is equal to the number of channels. \( \text{correl}[i] \) contains the auto-correlation of channel \( i \).
- **img**: Pointer to image.
- **dx**: Displacement in the X direction.
- **dy**: Displacement in the Y direction.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`mlib_ImageAutoCorrel(3MLIB)`, `mlib_ImageCrossCorrel(3MLIB)`, `mlib_ImageCrossCorrel_Fp(3MLIB)`, `attributes(5)`
The `mlib_ImageAve()` function computes the average of two images on a pixel-by-pixel basis. It uses the following equation:

\[ \text{dst}[x][y][i] = \frac{\text{src1}[x][y][i] + \text{src2}[x][y][i] + 1}{2} \]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See also `mlib_ImageAve_Fp(3MLIB), mlib_ImageAve_Fp_Inp(3MLIB), mlib_ImageAve_Inp(3MLIB), attributes(5)`
The `mlib_ImageAve_Fp()` function computes the average of two floating-point images on a pixel-by-pixel basis. It uses the following equation:

\[
\text{dst}[x][y][i] = \frac{(\text{src1}[x][y][i] + \text{src2}[x][y][i])}{2}
\]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageAve(3MLIB), mlib_ImageAve_Fp_Inp(3MLIB), mlib_ImageAve_Inp(3MLIB), attributes(5)`
The `mlib_ImageAve_Fp_Inp()` function computes the average of two floating-point images on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
src1dst[x][y][i] = \frac{src1dst[x][y][i] + src2[x][y][i]}{2}
\]

The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageAve(3MLIB), mlib_ImageAve_Fp(3MLIB), mlib_ImageAve_Inp(3MLIB), attributes(5)`
mlib_ImageAve_Inp(3MLIB)

**Name**
mlib_ImageAve_Inp – average of two images, in place

**Synopsis**
```
c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageAve_Inp(mlib_image *src1dst, const mlib_image *src2);
```

**Description**
The `mlib_ImageAve_Inp()` function computes the average of two images on a pixel-by-pixel basis, in place.

It uses the following equation:
```
src1dst[x][y][i] = (src1dst[x][y][i] + src2[x][y][i] + 1) / 2
```

**Parameters**
The function takes the following arguments:
- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageAve(3MLIB), mlib_ImageAve_Fp(3MLIB), mlib_ImageAve_Fp_Inp(3MLIB), attributes(5)
Name  mlib_ImageBlend1_Fp_Inp - blend with an alpha image

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlend1_Fp_Inp(mlib_image *src1dst,
    const mlib_image *src2, const mlib_image *alpha);

Description  The mlib_ImageBlend1_Fp_Inp() function blends two images together, in place, on a
pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The alpha image
can be a single-channel image or have the same number of channels as the source and
destination images.

It uses the following equation when the alpha image is a single-channel image:

\[
src1dst[x][y][i] = alpha[x][y][0] * src1dst[x][y][i] +
(1 - alpha[x][y][0]) * src2[x][y][i]
\]

It uses the following equation when the alpha image has the same number of channels as the
source and destination images:

\[
src1dst[x][y][i] = alpha[x][y][i] * src1dst[x][y][i] +
(1 - alpha[x][y][i]) * src2[x][y][i]
\]

Parameters  The function takes the following arguments:

src1dst  Pointer to first source and destination image.
src2  Pointer to second source image.
alpha  Alpha image used to control blending. The pixels in this image should have values
in the range of [0.0,1.0].

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageBlend(3MLIB), mlib_ImageBlend_Fp(3MLIB), mlib_ImageBlend1_Inp(3MLIB),
mlib_ImageBlend2_Fp_Inp(3MLIB), mlib_ImageBlend2_Inp(3MLIB), attributes(5)
The `mlib_ImageBlend1_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The alpha image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the alpha image is a single-channel image:

\[
src1dst[x][y][i] = a[x][y][0] \times src1dst[x][y][i] + (1 - a[x][y][0]) \times src2[x][y][i]
\]

It uses the following equation when the alpha image has the same number of channels as the source and destination images:

\[
src1dst[x][y][i] = a[x][y][i] \times src1dst[x][y][i] + (1 - a[x][y][i]) \times src2[x][y][i]
\]

The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.
- `alpha` Alpha image used to control blending. The a value equals \((\text{alpha} \times 2^{-8})\) for MLIB_BYTE image, \((\text{alpha} \times 2^{-15})\) for MLIB_SHORT image, \((\text{alpha} \times 2^{-16})\) for MLIB_USHORT image, and \((\text{alpha} \times 2^{-31})\) for MLIB_INT image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See Also `mlib_ImageBlend(3MLIB), mlib_ImageBlend_Fp(3MLIB), mlib_ImageBlend1_Fp_Inp(3MLIB), mlib_ImageBlend2_Fp_Inp(3MLIB), mlib_ImageBlend2_Inp(3MLIB), attributes(5)`
The `mlib_ImageBlend2_Fp_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when `a` is also on a pixel basis. The `alpha` image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the `alpha` image is a single-channel image:

\[
\text{src2dst}[x][y][i] = \text{alpha}[x][y][0]*\text{src1}[x][y][i] + \\
(1 - \text{alpha}[x][y][0])*\text{src2dst}[x][y][i]
\]

It uses the following equation when the `alpha` image has the same number of channels as the source and destination images:

\[
\text{src2dst}[x][y][i] = \text{alpha}[x][y][i]*\text{src1}[x][y][i] + \\
(1 - \text{alpha}[x][y][i])*\text{src2dst}[x][y][i]
\]

It uses the following equation:

**Parameters**

The function takes the following arguments:

- `src2dst` Pointer to second source and destination image.
- `src1` Pointer to first source image.
- `alpha` Alpha image used to control blending. The pixels in this image should have values in the range of \([0.0, 1.0]\).

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageBlend(3MLIB)`, `mlib_ImageBlend_Fp(3MLIB)`,
- `mlib_ImageBlend1_Fp_Inp(3MLIB)`, `mlib_ImageBlend1_Inp(3MLIB)`,
- `mlib_ImageBlend2_Inp(3MLIB)`
**Name**  
mlib_ImageBlend2_Inp – blend with an alpha image, in place

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

```c
mlib_status mlib_ImageBlend2_Inp(mlib_image *src2dst,
                        const mlib_image *src1, const mlib_image *alpha);
```

**Description**  
The `mlib_ImageBlend2_Inp()` function blends two images together, in place, on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

\[
src2dst[x][y][i] = a[x][y][0]*src1[x][y][i] + (1 - a[x][y][0])*src2dst[x][y][i]
\]

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

\[
src2dst[x][y][i] = a[x][y][i]*src1[x][y][i] + (1 - a[x][y][i])*src2dst[x][y][i]
\]

**Parameters**  
The function takes the following arguments:

- `src2dst`  
  Pointer to second source and destination image.

- `src1`  
  Pointer to first source image.

- `alpha`  
  Alpha image used to control blending. The a value equals \((\text{alpha} * 2^{(-8)})\) for MLIB_BYTE image, \((\text{alpha} * 2^{(-15)})\) for MLIB_SHORT image, \((\text{alpha} * 2^{(-16)})\) for MLIB_USHORT image, and \((\text{alpha} * 2^{(-31)})\) for MLIB_INT image.

**Return Values**  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageBlend(3MLIB), mlib_ImageBlend_Fp(3MLIB),  
mlib_ImageBlend1_Fp_Inp(3MLIB), mlib_ImageBlend1_Inp(3MLIB),  
mlib_ImageBlend2_Fp_Inp(3MLIB), attributes(5)
# mlib_ImageBlend

**mlib_ImageBlend** – blend with an alpha image

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlend(mlib_image *dst, const mlib_image *src1,
const mlib_image *src2, const mlib_image *alpha);
```

**Description**

The `mlib_ImageBlend()` function blends two images together on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The alpha image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the alpha image is a single-channel image:

\[
dst[x][y][i] = a[x][y][0]*src1[x][y][i] + \\
(1 - a[x][y][0])*src2[x][y][i]
\]

It uses the following equation when the alpha image has the same number of channels as the source and destination images:

\[
dst[x][y][i] = a[x][y][i]*src1[x][y][i] + \\
(1 - a[x][y][i])*src2[x][y][i]
\]

**Parameters**

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src1** Pointer to first source image.
- **src2** Pointer to second source image.
- **alpha** Alpha image used to control blending. The a value equals \(\alpha * 2^{**(-8)}\) for MLIB_BYTE image, \(\alpha * 2^{**(-16)}\) for MLIB_SHORT image, \(\alpha * 2^{**(-16)}\) for MLIB_UH银HORT image, and \(\alpha * 2^{**(-31)}\) for MLIB_INT image.

**Return Values**

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageBlend_Fp(3MLIB)`, `mlib_ImageBlend1_Fp_Inp(3MLIB)`, `mlib_ImageBlend1_Inp(3MLIB)`, `mlib_ImageBlend2_Fp_Inp(3MLIB)`, `mlib_ImageBlend2_Inp(3MLIB)`, attributes(5)
mlib_ImageBlend_BSRC1_BSRC2(3MLIB)

**Name**
mlib_ImageBlend_BSRC1_BSRC2, mlib_ImageBlend_DA_DA, mlib_ImageBlend_DA_DC,
mlib_ImageBlend_DA_OMDA, mlib_ImageBlend_DA_OMDC,
mlib_ImageBlend_DA_OMSA, mlib_ImageBlend_DA_ONE, mlib_ImageBlend_DA_SA,
mlib_ImageBlend_DA_SAS, mlib_ImageBlend_DA_ZERO, mlib_ImageBlend_OMDA_DA,
mlib_ImageBlend_OMDA_DC, mlib_ImageBlend_OMDA_OMDA,
mlib_ImageBlend_OMDA_OMDC, mlib_ImageBlend_OMDA_OMSA,
mlib_ImageBlend_OMDA_ONE, mlib_ImageBlend_OMDA_SA,
mlib_ImageBlend_OMDA_SAS, mlib_ImageBlend_OMDA_ZERO,
mlib_ImageBlend_OMSA_DA, mlib_ImageBlend_OMSA_DC,
mlib_ImageBlend_OMSA_OMDA, mlib_ImageBlend_OMSA_OMDC,
mlib_ImageBlend_OMSA_OMSA, mlib_ImageBlend_OMSA_OMSA_ONE,
mlib_ImageBlend_OMSA_SA, mlib_ImageBlend_OMSA_SAS,
mlib_ImageBlend_OMSA_ZERO, mlib_ImageBlend_OMSC_DA,
mlib_ImageBlend_OMSC_DC, mlib_ImageBlend_OMSC_OMDA,
mlib_ImageBlend_OMSC_OMDC, mlib_ImageBlend_OMSC_OMSA,
mlib_ImageBlend_OMSC_ONE, mlib_ImageBlend_OMSC_SA,
mlib_ImageBlend_OMSC_SAS, mlib_ImageBlend_OMSC_ZERO,
mlib_ImageBlend_ONE_DA, mlib_ImageBlend_ONE_DC,
mlib_ImageBlend_ONE_OMDA, mlib_ImageBlend_ONE_OMDC,
mlib_ImageBlend_ONE_OMSA, mlib_ImageBlend_ONE_ONE,
mlib_ImageBlend_ONE_SA, mlib_ImageBlend_ONE_SAS, mlib_ImageBlend_ONE_ZERO,
mlib_ImageBlend_SA_DA, mlib_ImageBlend_SA_DC, mlib_ImageBlend_SA_OMDA,
mlib_ImageBlend_SA_OMDC, mlib_ImageBlend_SA_OMSA, mlib_ImageBlend_SA_ONE,
mlib_ImageBlend_SA_SA, mlib_ImageBlend_SA_SAS, mlib_ImageBlend_SA_ZERO,
mlib_ImageBlend_SC_DA, mlib_ImageBlend_SC_DC, mlib_ImageBlend_SC_OMDA,
mlib_ImageBlend_SC_OMDC, mlib_ImageBlend_SC_OMSA, mlib_ImageBlend_SC_ONE,
mlib_ImageBlend_SC_SA, mlib_ImageBlend_SC_SAS, mlib_ImageBlend_SC_ZERO,
mlib_ImageBlend_ZERO_DA, mlib_ImageBlend_ZERO_DC,
mlib_ImageBlend_ZERO_OMDA, mlib_ImageBlend_ZERO_OMDC,
mlib_ImageBlend_ZERO_OMSA, mlib_ImageBlend_ZERO_ONE,
mlib_ImageBlend_ZERO_SA, mlib_ImageBlend_ZERO_SAS,
mlib_ImageBlend_ZERO_ZERO – blending

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlend_BSRC1_BSRC2(mlib_image *dst,
    const mlib_image *src1, const mlib_image *src2, mlib_s32 cmask);

**Description**
This group of functions supports digital image composition. They are low-level, non-in-place, blending functions.

The image type must be MLIB_BYTE. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.
BSRC1 is one of the following: ZERO, ONE, SC, OMSC, DA, SA, OMDA, or OMSA. BSRC2 is one of the following: ZERO, ONE, DC, OMDC, DA, SA, OMDA, OMSA, or SAS.

The following are predefined blend factor types used in mediaLib image composition functions.

/* image blend factors */
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mlib_blend;

See the following table for the definitions of the blend factors.

<table>
<thead>
<tr>
<th>Type</th>
<th>Blend Factor[*]</th>
<th>Abbr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BLEND_ZERO</td>
<td>(0,0,0,0)</td>
<td>ZERO</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE</td>
<td>(1,1,1,1)</td>
<td>ONE</td>
</tr>
<tr>
<td>MLIB_BLEND_DST_COLOR</td>
<td>(Rd,Gd,Bd,Ad)</td>
<td>DC</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_COLOR</td>
<td>(Rs,Gs,Bs,As)</td>
<td>SC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_DST_COLOR</td>
<td>(1,1,1,1)-(Rd,Gd,Bd,Ad)</td>
<td>OMDC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_SRC_COLOR</td>
<td>(1,1,1,1)-(Rs,Gs,Bs,As)</td>
<td>OMSC</td>
</tr>
<tr>
<td>MLIB_BLEND_DST_ALPHA</td>
<td>(Ad,Ad,Ad,Ad)</td>
<td>DA</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_ALPHA</td>
<td>(As,As,As,As)</td>
<td>SA</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_DST_ALPHA</td>
<td>(1,1,1,1)-(Ad,Ad,Ad,Ad)</td>
<td>OMDA</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_SRC_ALPHA</td>
<td>(1,1,1,1)-(As,As,As,As)</td>
<td>OMSA</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_ALPHA_SATURATE</td>
<td>(f,f,f,1)</td>
<td>SAS</td>
</tr>
</tbody>
</table>

[*]: The components of the first source image pixel are (Rd,Gd,Bd,Ad), and the components of the second source pixel are (Rs,Gs,Bs,As). Function \( f = \min(As,1-Ad) \).

The blending formula for non-in-place processing is:
\[ Cd = Cs1*S1 + Cs2*S2 \]

where \( Cd \) is the destination pixel \((R_d,G_d,B_d, A_d)\), \( Cs1 \) is the first source pixel \((R_{s1},G_{s1},B_{s1},A_{s1})\), \( Cs2 \) is the second source pixel \((R_{s2},G_{s2},B_{s2},A_{s2})\), and \( S1 \) and \( S2 \) are the blend factors for the first and second sources, respectively.

**Parameters**

Each of the functions takes the following arguments:

- \( dst \) Pointer to destination image.
- \( src1 \) Pointer to the first source image.
- \( src2 \) Pointer to the second source image.
- \( cmask \) Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. \( cmask \) must be either 0x01 or 0x08.

**Return Values**

Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

**Attributes**

See attributes\((5)\) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

\texttt{mlib\_ImageBlend\_BSRC1\_BSRC2(3MLIB)}, \texttt{mlib\_ImageComposite(3MLIB)}, \texttt{mlib\_ImageComposite\_Inp(3MLIB)}, attributes\((5)\)
Name

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlend_BSRC1_BSRC2_Inp(mlib_image *srcdst,
const mlib_image *src2, mlib_s32 cmask);
This group of functions supports digital image composition. They are low-level, in-place, blending functions.

The image type must be \texttt{MLIB\_BYTE}. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.

\texttt{BSRC1} is one of the following: ZERO, ONE, SC, OMSC, DA, SA, OMDA, or OMSA. \texttt{BSRC2} is one of the following: ZERO, ONE, DC, OMDC, DA, SA, OMDA, OMSA, or SAS.

The following are predefined blend factor types used in mediaLib image composition functions.

```c
/* image blend factors */
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mlib_blend;
```

See the following table for the definitions of the blend factors.

<table>
<thead>
<tr>
<th>Type</th>
<th>Blend Factor [*]</th>
<th>Abbr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BLEND_ZERO</td>
<td>(0,0,0,0)</td>
<td>ZERO</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE</td>
<td>(1,1,1,1)</td>
<td>ONE</td>
</tr>
<tr>
<td>MLIB_BLEND_DST_COLOR</td>
<td>(Rd,Gd,Bd,Ad)</td>
<td>DC</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_COLOR</td>
<td>(Rs,Gs, Bs, As)</td>
<td>SC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_DST_COLOR</td>
<td>(1,1,1,1)-(Rd,Gd,Bd,Ad)</td>
<td>OMDC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_SRC_COLOR</td>
<td>(1,1,1,1)-(Rs,Gs, Bs, As)</td>
<td>OMSC</td>
</tr>
<tr>
<td>MLIB_BLEND_DST_ALPHA</td>
<td>(Ad,Ad,Ad,Ad)</td>
<td>DA</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_ALPHA</td>
<td>(As,As,As,As)</td>
<td>SA</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_DST_ALPHA</td>
<td>(1,1,1,1)-(Ad,Ad,Ad,Ad)</td>
<td>OMDA</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_SRC_ALPHA</td>
<td>(1,1,1,1)-(As,As,As,As)</td>
<td>OMSA</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_ALPHA_SATURATE</td>
<td>(fff,1)</td>
<td>SAS</td>
</tr>
</tbody>
</table>
The componentsof the first source image pixel are \((R_d, G_d, B_d, A_d)\), and the components of the second source pixel are \((R_s, G_s, B_s, A_s)\). Function \(f = \min(A_s, 1 - A_d)\). The first source image is also the destination image.

The blending formula for in-place processing is:

\[
C_d = C_d \cdot D + C_s \cdot S
\]

where \(C_d\) is the destination pixel \((R_d, G_d, B_d, A_d)\), \(C_s\) is the source pixel \((R_s, G_s, B_s, A_s)\), and \(D\) and \(S\) are the blend factors for the destination and source, respectively.

**Parameters**

Each of the functions takes the following arguments:

- \(src1dst\) Pointer to the first source and the destination image.
- \(src2\) Pointer to the second source image.
- \(cmask\) Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. \(cmask\) must be either \(0x01\) or \(0x08\).

**Return Values**

Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

\texttt{mlib\_ImageBlend\_BSRC1\_BSRC2(3MLIB)}, \texttt{mlib\_ImageComposite(3MLIB)}, \texttt{mlib\_ImageComposite\_Inp(3MLIB)}, attributes(5)
### mlib_ImageBlendColor(3MLIB)

**Name**
mlib_ImageBlendColor – blend an image and a color

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlendColor(mlib_image *dst, const mlib_image *src, const mlib_s32 *color, mlib_s32 cmask);
```

**Description**
The `mlib_ImageBlendColor()` function blends an image and a color with the alpha channel.

It uses the following equation:
\[
C_d = C_s \times A_s + C_c \times (1 - A_s)
\]
\[
A_d = 1.0
\]

where, \(C_s\) and \(C_d\) are the RGB color components of the source and destination images, respectively. \(A_s\) and \(A_d\) are the alpha components of the source and destination images, respectively. \(C_c\) is the color component of the constant color.

For `MLIB_BYTE` images, the alpha coefficients are in Q8 format. For `MLIB_SHORT` images, the alpha coefficients are in Q15 format and must be positive. For `MLIB_USHORT` images, the alpha coefficients are in Q16 format. For `MLIB_INT` images, the alpha coefficients are in Q31 format and must be positive.

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

**Parameters**
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `color` Array of constant color components.
- `cmask` Channel mask to indicate the alpha channel. Each bit of `cmask` represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also

mlib_ImageBlendColor_Inp(3MLIB), mlib_ImageBlendColor_Fp(3MLIB),
mlib_ImageBlendColor_Fp_Inp(3MLIB), attributes(5)
mlib_imageBlendColor_Fp() function blends an image and a color with the alpha channel. It uses the following equation:

\[ \begin{align*}
    C_d &= C_s * A_s + C_c * (1 - A_s) \\
    A_d &= 1.0
\end{align*} \]

where, \( C_s \) and \( C_d \) are the RGB color components of the source and destination images, respectively. \( A_s \) and \( A_d \) are the alpha components of the source and destination images, respectively. \( C_c \) is the color component of the constant color.

For MLIB_FLOAT and MLIB_DOUBLE images, the alpha coefficients are assumed to be in the range of \([0.0, 1.0]\).

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

The function takes the following arguments:

- \( dst \) Pointer to destination image.
- \( src \) Pointer to source image.
- \( color \) Array of constant color components.
- \( cmask \) Channel mask to indicate the alpha channel. Each bit of cmask represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageBlendColor_Fp(3MLIB), mlib_ImageBlendColor(3MLIB),
         mlib_ImageBlendColor_Inp(3MLIB), attributes(5)
The `mlib_ImageBlendColor_Fp_Inp()` function blends an image and a color with the alpha channel.

It uses the following equation:

\[
C_d = C_s \cdot A_s + C_c \cdot (1 - A_s)
\]

\[
A_d = 1.0
\]

where, \(C_s\) and \(C_d\) are the RGB color components of the source and destination images, respectively. \(A_s\) and \(A_d\) are the alpha components of the source and destination images, respectively. \(C_c\) is the color component of the constant color.

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the alpha coefficients are assumed to be in the range of \([0.0, 1.0]\).

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

### Parameters

- **srcdst** Pointer to the source and destination image.
- **color** Array of constant color components.
- **cmask** Channel mask to indicate the alpha channel. Each bit of cmask represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageBlendColor_Fp(3MLIB)`
- `mlib_ImageBlendColor(3MLIB)`
- `mlib_ImageBlendColor_Inp(3MLIB)`
- attributes(5)
The `mlib_ImageBlendColor_Inp()` function blends an image and a color with the alpha channel.

It uses the following equation:

\[
Cd = Cs \times As + Cc \times (1 - As) \\
Ad = 1.0
\]

where, \(Cs\) and \(Cd\) are the RGB color components of the source and destination images, respectively. \(As\) and \(Ad\) are the alpha components of the source and destination images, respectively. \(Cc\) is the color component of the constant color.

For `MLIB_BYTE` images, the alpha coefficients are in Q8 format. For `MLIB_SHORT` images, the alpha coefficients are in Q15 format and must be positive. For `MLIB_USHORT` images, the alpha coefficients are in Q16 format. For `MLIB_INT` images, the alpha coefficients are in Q31 format and must be positive.

The images can have two to four channels. The length of color array must not be less than the number of channels in the images.

### Parameters

The function takes the following arguments:

- `srcdst` Pointer to the source and destination image.
- `color` Array of constant color components.
- `cmask` Channel mask to indicate the alpha channel. Each bit of `cmask` represents a channel in the image. The channel corresponding to the highest bit with value 1 is the alpha channel.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageBlendColor(3MLIB), mlib_ImageBlendColor_Fp(3MLIB),
mlib_ImageBlendColor_Fp_Inp(3MLIB), attributes(5)
mlib_ImageBlend_Fp – blend with an alpha image

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageBlend_Fp(mlib_image *dst, const mlib_image *src1, const mlib_image *src2, const mlib_image *alpha);
```

**Description**

The `mlib_ImageBlend_Fp()` function blends two images together on a pixel-by-pixel basis using an alpha image, when alpha is also on a pixel basis. The *alpha* image can be a single-channel image or have the same number of channels as the source and destination images.

It uses the following equation when the *alpha* image is a single-channel image:

\[
dst[x][y][i] = \alpha[x][y][0]*src1[x][y][i] + (1 - \alpha[x][y][0])*src2[x][y][i]
\]

It uses the following equation when the *alpha* image has the same number of channels as the source and destination images:

\[
dst[x][y][i] = \alpha[x][y][i]*src1[x][y][i] + (1 - \alpha[x][y][i])*src2[x][y][i]
\]

**Parameters**

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src1** Pointer to first source image.
- **src2** Pointer to second source image.
- **alpha** Alpha image used to control blending. The pixels in this image should have values in the range of [0.0, 1.0].

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageBlend(3MLIB)`, `mlib_ImageBlend1_Fp_Inp(3MLIB)`, `mlib_ImageBlend1_Inp(3MLIB)`, `mlib_ImageBlend2_Fp_Inp(3MLIB)`, `mlib_ImageBlend2_Inp(3MLIB)`, `attributes(5)`
mlib_ImageBlendMulti – blend multiple images

#include <mlib.h>

mlib_status mlib_ImageBlendMulti(mlib_image *dst, const mlib_image **srcs,
const mlib_image **alphas, const mlib_s32 *c, mlib_s32 n);

The mlib_ImageBlendMulti() function blends multiple source images, using multiple alpha images, into a single destination image.

All images involved should have the same data type and same size and the source and destination images should have the same number of channels. The alpha images should have either 1 channel or the same number of channels as the sources and destination. A single-channel alpha image would be applied to all channels of the corresponding source image. Single and multi-channel alpha images can be mixed in the same invocation.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][i] &= \frac{\sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][j] * \text{srcs}[k][x][y][i]\}}{\sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][j]\}} \\
\text{or}
\text{dst}[x][y][i] &= \text{c}[i] \quad \text{if } \sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][j]\} = 0
\end{align*}
\]

where \(j = i\) for multi-channel alpha images; \(j = 0\) for single-channel alpha images.

The function takes the following arguments:

- \(dst\) Pointer to destination image.
- \(srcs\) Pointer to an array of source images.
- \(alphas\) Pointer to an array of alpha images.
- \(c\) Background color.
- \(n\) Number of source images to be blended.

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageBlendMulti_Fp(3MLIB), mlib_ImageBlend(3MLIB), mlib_ImageBlend_Fp(3MLIB), attributes(5)
The `mlib_ImageBlendMulti_Fp()` function blends multiple source images, using multiple alpha images, into a single destination image.

All images involved should have the same data type and same size and the source and destination images should have the same number of channels. The alpha images should have either 1 channel or the same number of channels as the sources and destination. A single-channel alpha image would be applied to all channels of the corresponding source image. Single and multi-channel alpha images can be mixed in the same invocation.

It uses the following equation:

\[
\text{dst}[x][y][i] = \frac{\sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][0] \times \text{srcs}[k][x][y][i]\}}{\sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][0]\}}
\]

or

\[
\text{dst}[x][y][i] = c[i] \quad \text{if} \sum_{k=0}^{n-1} \{\text{alphas}[k][x][y][0]\} = 0
\]

where \( j = i \) for multi-channel alpha images; \( j = 0 \) for single-channel alpha images.

### Parameters

- **dst** Pointer to destination image.
- **srcs** Pointer to an array of source images.
- **alphas** Pointer to an array of alpha images.
- **c** Background color.
- **n** Number of source images to be blended.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`. 
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageBlendMulti(3MLIB), mlib_ImageBlend(3MLIB), mlib_ImageBlend_Fp(3MLIB), attributes(5)
The `mlib_ImageBlendRGBA2ARGB()` function blends the source image of the RGBA format into the destination image of the ARGB format.

The image type must be `MLIB_BYTE`. The source and destination images must contain four channels.

It uses the following equation:

\[
Cd = Cs \times As + Cd \times (1 - As) \\
Ad = Ad
\]

where, \( Cs \) and \( Cd \) are the RGB color components of the source and destination images, respectively. \( As \) and \( Ad \) are the alpha components of the source and destination images, respectively.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageBlendRGBA2BGRA(3MLIB)`, `mlib_ImageBlend_OMSA_SA_Inp(3MLIB)`, `attributes(5)`
#include <mlib.h>

mlib_status mlib_ImageBlendRGBA2BGRA(mlib_image *dst, const mlib_image *src);

The `mlib_ImageBlendRGBA2BGRA()` function blends the source image of the RGBA format into the destination image of the BGRA format. The image type must be MLIB_BYTE. The source and destination images must contain four channels.

It uses the following equation:

\[ C_d = C_s \times A_s + C_d \times (1 - A_s) \]
\[ A_d = A_d \]

where, \( C_s \) and \( C_d \) are the RGB color components of the source and destination images, respectively. \( A_s \) and \( A_d \) are the alpha components of the source and destination images, respectively.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageBlendRGBA2ARGB(3MLIB), mlib_ImageBlend_OMSA_SA_Inp(3MLIB), attributes(5)
#include <mlib.h>

mlib_status mlib_ImageChannelCopy(mlib_image *dst, const mlib_image *src, mlib_s32 cmask);

The `mlib_ImageChannelCopy()` function copies the selected channels of the source image into the corresponding channels of the destination image. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

**Parameters**
- `dst`  Pointer to a destination image.
- `src`  Pointer to a source image.
- `cmask`  Source or destination channel selection mask. Each bit of the mask represents a channel in the image data. The least significant bit (LSB) of the mask corresponds to the last channel in the image data. A bit with a value of 1 indicates that the channel is selected.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageChannelExtract(3MLIB)`, `mlib_ImageChannelInsert(3MLIB)`,
- `mlib_ImageChannelMerge(3MLIB)`, `mlib_ImageChannelSplit(3MLIB)`, attributes(5)
In the `mlib_ImageChannelExtract()` function, the selected N channels in the source image are copied into the destination image, where N is the number of channels in the destination image. If more than N channels are selected, then the leftmost N channels are extracted. If less than N channels are selected, then the function returns failure status. The channel mask is defined with respect to the source image. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

The function takes the following arguments:

- `dst` Pointer to a destination image.
- `src` Pointer to a source image.
- `cmask` Source or destination channel selection mask. Each bit of the mask represents a channel in the image data. The least significant bit (LSB) of the mask corresponds to the last channel in the image data. A bit with a value of 1 indicates that the channel is selected.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageChannelCopy(3MLIB)`, `mlib_ImageChannelInsert(3MLIB)`, `mlib_ImageChannelMerge(3MLIB)`, `mlib_ImageChannelSplit(3MLIB)`, `attributes(5)`
mlib_ImageChannelInsert — channel insert

Name  
mlib_ImageChannelInsert – channel insert

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageChannelInsert(mlib_image *dst, const mlib_image *src,
                        mlib_s32 cmask);

Description  
In the mlib_ImageChannelInsert() function, all N channels in the source image are copied into the selected channels in the destination image, where N is the number of channels in the source image. If more than N channels are selected, then the leftmost N channels are inserted. If less than N channels are selected, then the function returns failure status. The channel mask is defined with respect to the destination image. The data type of the image can be MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters  
The function takes the following arguments:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>Pointer to a destination image.</td>
</tr>
<tr>
<td>src</td>
<td>Pointer to a source image.</td>
</tr>
<tr>
<td>cmask</td>
<td>Source or destination channel selection mask. Each bit of the mask represents a channel in the image data. The least significant bit (LSB) of the mask corresponds to the last channel in the image data. A bit with a value of 1 indicates that the channel is selected.</td>
</tr>
</tbody>
</table>

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageChannelCopy(3MLIB), mlib_ImageChannelExtract(3MLIB), mlib_ImageChannelMerge(3MLIB), mlib_ImageChannelSplit(3MLIB), attributes(5)
mlib_ImageChannelMerge

Name       mlib_ImageChannelMerge – channel merge

Synopsis   cc [ flag... ] file... -lmlib [ library... ]
            #include <mlib.h>
            
            mlib_status mlib_ImageChannelMerge(mlib_image *dst,
                                                   const mlib_image **srcs);

Description The mlib_ImageChannelMerge() function converts an array of single-channel images into a multi-channel image.

A0 A1 A2 ...
B0 B1 B2 ... ===> A0 B0 C0 A1 B1 C1 A2 B2 C2 ...
C0 C1 C2 ...

All images must have the same type, same width, and same height. The data type of the images can be MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE. The destination image must have the number of channels equal to the number of images in the srcs array. The source images must be single-channel images.

Parameters  The function takes the following arguments:

dst         Pointer to a multi-channel destination image.

srcs        Pointer to an array of single-channel source images.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

+---------------------------------+------------------+
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageChannelCopy(3MLIB), mlib_ImageChannelExtract(3MLIB),
          mlib_ImageChannelInsert(3MLIB), mlib_ImageChannelSplit(3MLIB), attributes(5)
The `mlib_ImageChannelSplit()` function converts a multi-channel image into an array of single-channel images.

```
A0 A1 A2 ...
A0 B0 C0 A1 B1 C1 A2 B2 C2 ... ===> B0 B1 B2 ...
C0 C1 C2 ...
```

All images must have the same type, same width, and same height. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`. The source image must have the number of channels equal to the number of images in the `dsts` array. The destination images must be single-channel images.

**Parameters**
The function takes the following arguments:

- `dsts` Pointer to an array of single-channel destination images.
- `src` Pointer to a multi-channel source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageChannelCopy(3MLIB)`, `mlib_ImageChannelExtract(3MLIB)`, `mlib_ImageChannelInsert(3MLIB)`, `mlib_ImageChannelMerge(3MLIB)`, `attributes(5)`
mlib_ImageClear() function sets an image to a specific color. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

The function takes the following arguments:

- `img` Pointer to an image.
- `color` Array of color values by channel. `color[i]` contains the value for channel i.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_ImageClear_Fp(3MLIB), mlib_ImageClearEdge(3MLIB), mlib_ImageClearEdge_Fp(3MLIB), attributes(5)
### Name
mlib_ImageClearEdge – sets edges of an image to a specific color

#### Synopsis
```
cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageClearEdge(mlib_image *img, mlib_s32 dx, mlib_s32 dy, const mlib_s32 *color);
```

#### Description
The `mlib_ImageClearEdge()` function sets edges of an image to a specific color. This function can be used in conjunction with the convolve and other spatial functions to fill in the pixel values along the edges. The data type of the image can be `MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT`.

#### Parameters
The function takes the following arguments:
- **img**: Pointer to an image.
- **dx**: Number of columns on the left and right edges of the image to be cleared.
- **dy**: Number of rows at the top and bottom edges of the image to be cleared.
- **color**: Array of color values by channel. `color[i]` contains the value for channel i.

#### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

#### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

#### See Also
- `mlib_ImageClear(3MLIB), mlib_ImageClear_Fp(3MLIB), mlib_ImageClearEdge_Fp(3MLIB), attributes(5)`
The `mlib_ImageClearEdge_Fp()` function sets edges of an image to a specific color. This function can be used in conjunction with the `convolve` and other spatial functions to fill in the pixel values along the edges. The data type of the image can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The function takes the following arguments:

- **img**: Pointer to an image.
- **dx**: Number of columns on the left and right edges of the image to be cleared.
- **dy**: Number of rows at the top and bottom edges of the image to be cleared.
- **color**: Array of color values by channel. `color[i]` contains the value for channel `i`.

The function returns `MLIB_SUCCESS` if successful. Otherwise, it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageClear(3MLIB)`, `mlib_ImageClearEdge(3MLIB)`, `mlib_ImageClearEdge_Fp(3MLIB)`, `attributes(5)`
# mlib_ImageClear_Fp

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageClear_Fp(mlib_image *img, const mlib_d64 *color);
```

## Description

The `mlib_ImageClear_Fp()` function sets an image to a specific color. The data type of the image can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

It uses the following equation:

```
img[x][y][i] = color[i]
```

## Parameters

- **img**: Pointer to an image.
- **color**: Array of color values by channel. `color[i]` contains the value for channel `i`.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_ImageClear(3MLIB)`, `mlib_ImageClearEdge(3MLIB)`, `mlib_ImageClearEdge_Fp(3MLIB)`, `attributes(5)`
Name  mlib_ImageColorConvert1 – color conversion using a 3x3 floating-point matrix

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorConvert1(mlib_image *dst, const mlib_image *src,
const mlib_d64 *cmat);

Description  The mlib_ImageColorConvert1() function takes a 3x3 floating-point conversion matrix and
converts the source color image to the destination color image.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][0] &= cmat[0] \cdot \text{src}[x][y][0] + cmat[1] \cdot \text{src}[x][y][1] + cmat[2] \cdot \text{src}[x][y][2] \\
\text{dst}[x][y][1] &= cmat[3] \cdot \text{src}[x][y][0] + cmat[4] \cdot \text{src}[x][y][1] + cmat[5] \cdot \text{src}[x][y][2] \\
\text{dst}[x][y][2] &= cmat[6] \cdot \text{src}[x][y][0] + cmat[7] \cdot \text{src}[x][y][1] + cmat[8] \cdot \text{src}[x][y][2]
\end{align*}
\]

Parameters  The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **cmat** Conversion matrix in row major order.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageColorConvert1_Fp(3MLIB), mlib_ImageColorConvert2(3MLIB),
mlib_ImageColorConvert2_Fp(3MLIB), mlib_ImageColorRGB2XYZ(3MLIB),
mlib_ImageColorRGB2XYZ_Fp(3MLIB), mlib_ImageColorXYZ2RGB(3MLIB),
mlib_ImageColorXYZ2RGB_Fp(3MLIB), attributes(5)
The `mlib_ImageColorConvert1_Fp()` function takes a 3x3 floating point conversion matrix and converts the floating point source color image to the destination color image.

It uses the following equation:

$$
\begin{align*}
\text{dst}[x][y][0] &= \text{cmat}[0] \cdot \text{src}[x][y][0] + \text{cmat}[1] \cdot \text{src}[x][y][1] + \text{cmat}[2] \\
\text{dst}[x][y][1] &= \text{cmat}[3] \cdot \text{src}[x][y][0] + \text{cmat}[4] \cdot \text{src}[x][y][1] + \text{cmat}[5] \\
\text{dst}[x][y][2] &= \text{cmat}[6] \cdot \text{src}[x][y][0] + \text{cmat}[7] \cdot \text{src}[x][y][1] + \text{cmat}[8]
\end{align*}
$$

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `cmat` Conversion matrix in row major order.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageColorConvert1(3MLIB), mlib_ImageColorConvert2(3MLIB), mlib_ImageColorConvert2_Fp(3MLIB), mlib_ImageColorRGB2XYZ(3MLIB), mlib_ImageColorRGB2XYZ_Fp(3MLIB), mlib_ImageColorXYZ2RGB(3MLIB), mlib_ImageColorXYZ2RGB_Fp(3MLIB), attributes(5)`
The `mlib_ImageColorConvert2()` function takes a 3x3 floating-point conversion matrix and a three-element offset and converts the source color image to the destination color image. The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][0] &= \text{cmat}[0] \cdot \text{src}[x][y][0] + \text{offset}[0] \\
\text{dst}[x][y][1] &= \text{cmat}[3] \cdot \text{cmat}[4] \cdot \text{cmat}[5] \cdot \text{src}[x][y][1] + \text{offset}[1] \\
\text{dst}[x][y][2] &= \text{cmat}[6] \cdot \text{cmat}[7] \cdot \text{cmat}[8] \cdot \text{src}[x][y][2] + \text{offset}[2]
\end{align*}
\]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `cmat` Conversion matrix in row major order.
- `offset` Offset array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageColorConvert1(3MLIB)`, `mlib_ImageColorConvert1_Fp(3MLIB)`
- `mlib_ImageColorConvert2_Fp(3MLIB)`, `mlib_ImageColorRGB2YCC(3MLIB)`
- `mlib_ImageColorRGB2YCC_Fp(3MLIB)`, `mlib_ImageColorYCC2RGB(3MLIB)`
- `mlib_ImageColorYCC2RGB_Fp(3MLIB)` attributes(5)
mlib_ImageColorConvert2_Fp – color conversion using a 3x3 floating-point matrix and a three-element offset

Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageColorConvert2_Fp(mlib_image *dst,
 const mlib_image *src, const mlib_d64 *cmat,
 const mlib_d64 *offset);
```

Description

The `mlib_ImageColorConvert2_Fp()` function takes a 3x3 floating-point conversion matrix and a three-element offset and converts the floating-point source color image to the destination color image.

The source and destination images must be three-channel images.

It uses the following equation:

\[
|\text{dst}[x][y][0]| = |\text{cmat}[0] \text{ cmat}[1] \text{ cmat}[2]| * |\text{src}[x][y][0]| + |\text{offset}[0]|
|\text{dst}[x][y][1]| = |\text{cmat}[3] \text{ cmat}[4] \text{ cmat}[5]| * |\text{src}[x][y][1]| + |\text{offset}[1]|
|\text{dst}[x][y][2]| = |\text{cmat}[6] \text{ cmat}[7] \text{ cmat}[8]| * |\text{src}[x][y][2]| + |\text{offset}[2]|
\]

Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `cmat` Conversion matrix in row major order.
- `offset` Offset array.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageColorConvert1(3MLIB)`
- `mlib_ImageColorConvert1_Fp(3MLIB)`
- `mlib_ImageColorConvert2(3MLIB)`
- `mlib_ImageColorRGB2YCC(3MLIB)`
- `mlib_ImageColorRGB2YCC_Fp(3MLIB)`
- `mlib_ImageColorYCC2RGB(3MLIB)`
- `mlib_ImageColorYCC2RGB_Fp(3MLIB)`, `attributes(5)`
Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_ImageColorDitherFree(void * colormap);

Description

The `mlib_ImageColorDitherFree()` function releases an internal data structure, colormap, which was created by `mlib_ImageColorDitherInit()` and was used by one of the following functions for image dithering:

- `mlib_ImageColorErrorDiffusion3x3`
- `mlib_ImageColorErrorDiffusionMxN`
- `mlib_ImageColorOrderedDither8x8`
- `mlib_ImageColorOrderedDitherMxN`

Parameters

The function takes the following arguments:

- `colormap` Internal data structure for image dithering.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageColorDitherInit(3MLIB)`, `mlib_ImageColorErrorDiffusion3x3(3MLIB)`,
- `mlib_ImageColorErrorDiffusionMxN(3MLIB)`,
- `mlib_ImageColorOrderedDither8x8(3MLIB)`,
- `mlib_ImageColorOrderedDitherMxN(3MLIB)`, attributes(5)
mlib_ImageColorDitherInit – initialization for image dithering

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_ImageColorDitherInit(void **colormap, const mlib_s32 *dimensions,
mlib_type intype, mlib_type outtype, mlib_s32 channels, mlib_s32 entries,
mlib_s32 offset, void **lut);

Description

The mlib_ImageColorDitherInit() function creates an internal data structure, colormap, which can be used by one of the following functions for image dithering:

mlib_ImageColorErrorDiffusion3x3
mlib_ImageColorErrorDiffusionMxN
mlib_ImageColorOrderedDither8x8
mlib_ImageColorOrderedDitherMxN

The lut might have either 1 or 3 channels. The type of the lut can be one of the following:

MLIB_BYTE in, MLIB_BYTE out (i.e., BYTE-to-BYTE)
MLIB_BIT in, MLIB_BYTE out (i.e., BIT-to-BYTE)

If dimensions == NULL, then no colorcube will be created. In this case, the user-provided lookup table, lut, will be used for dithering.

If dimensions != NULL, then a colorcube is created from scratch in a way shown in the following example.

To dither an RGB image of type MLIB_BYTE to a color-indexed image of type MLIB_BYTE, we can use the following parameters:

mlib_s32 dimensions[] = {2, 3, 4};
mlib_type intype = MLIB_BYTE;
mlib_type outtype = MLIB_BYTE;
mlib_s32 channels = 3;
mlib_s32 offset = 6;

These values would lead to the creation of a colorcube that would dither red values in the source image to one of 2 red levels, green values to one of 3 green levels, and blue values to one of 4 blue levels. You could picture this colorcube as a cube with dimensions of 2, 3, and 4. The index values assigned to the elements in that cube can be described by the following lookup table:

<table>
<thead>
<tr>
<th>Indexes</th>
<th>Red Values</th>
<th>Green Values</th>
<th>Blue Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Indexes</td>
<td>Red Values</td>
<td>Green Values</td>
<td>Blue Values</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>255</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>85</td>
</tr>
<tr>
<td>13</td>
<td>255</td>
<td>0</td>
<td>85</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>128</td>
<td>85</td>
</tr>
<tr>
<td>15</td>
<td>255</td>
<td>128</td>
<td>85</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>255</td>
<td>85</td>
</tr>
<tr>
<td>17</td>
<td>255</td>
<td>255</td>
<td>85</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>19</td>
<td>255</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>128</td>
<td>170</td>
</tr>
<tr>
<td>21</td>
<td>255</td>
<td>128</td>
<td>170</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>255</td>
<td>170</td>
</tr>
<tr>
<td>23</td>
<td>255</td>
<td>255</td>
<td>170</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>25</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>128</td>
<td>255</td>
</tr>
<tr>
<td>27</td>
<td>255</td>
<td>128</td>
<td>255</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>29</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
</tbody>
</table>

The distance between level changes in each channel of the lookup table is determined by the following formulas:
multipliers[0] = signof(dimensions[0])*1;
multipliers[i] = signof(dimensions[i])*abs(multipliers[i-1]*dimension[i-1]);

A negative dimensions[i], so as to a negative multipliers[i], indicates that the values in a
color ramp for channel i should appear in decreasing as opposed to increasing order.

For each channel i, the values of the levels are determined by the following formulas:

double delta = (dataMax - dataMin)/(abs(dimensions[i]) - 1);
int levels[j] = (int)(j*delta + 0.5);

where dataMax and dataMin are the maximum and minimum values, respectively, for data

type intype.

Whenever a colorcube is created, if lut != NULL, the lookup table will be filled according to
the colorcube and supplied parameters like offset. For the example shown above, the lookup

table will start from line 6. In this case, it is the user's responsibility to allocate memory for the
lookup table.

Parameters The function takes the following arguments:

colormap Internal data structure for image dithering.
dimensions Dimensions of the colorcube in the colormap structure.
intype Data type of the source image and the lookup table.
outtype Data type of the destination indexed image.
channels Number of channels of the lookup table and source image.
entries Number of entries of the lookup table.
offset Index offset of the lookup table.
lut Lookup table.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageColorDitherFree(3MLIB), mlib_ImageColorErrorDiffusion3x3(3MLIB),
mlib_ImageColorErrorDiffusionMxN(3MLIB),
mlib_ImageColorOrderedDither8x8(3MLIB),
mlib_ImageColorOrderedDitherMxN(3MLIB), attributes(5)
mlib_ImageColorErrorDiffusion3x3(3MLIB)

Name
mlib_ImageColorErrorDiffusion3x3 – true color to indexed color conversion using error diffusion

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorErrorDiffusion3x3(mlib_image *dst,
const mlib_image *src, const mlib_s32 *kernel, mlib_s32 scale,
const void *colormap);

Description
The mlib_ImageColorErrorDiffusion3x3() function converts a true color image to a
pseudo color image with the method of error diffusion dithering. The source image can be an
MLIB_BYTE or MLIB_SHORT image with three or four channels. The destination must be a
single-channel MLIB_BYTE or MLIB_SHORT image.

The last parameter, colormap, is an internal data structure for inverse color mapping. Create it
by calling the mlib_ImageColorTrue2IndexInit() function.

Parameters
The function takes the following arguments:

dst Pointer to destination or destination image.
src Pointer to source or source image.
kernel Pointer to the 3x3 error-distribution kernel, in row major order.
scale The scaling factor for kernel to convert the input integer coefficients into
floating-point coefficients:
floating-point coefficient = integer coefficient * \n2**(-scale)

colormap Internal data structure for inverse color mapping.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageColorOrderedDither8x8(3MLIB), mlib_ImageColorTrue2Index(3MLIB),
mlib_ImageColorTrue2IndexFree(3MLIB), mlib_ImageColorTrue2IndexInit(3MLIB),
attributes(5)
**mlib/ImageColorErrorDiffusionMxN – true-color to indexed-color or grayscale to black-white conversion, using error diffusion**

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_ImageColorErrorDiffusionMxN(mlib_image *dst,
       const mlib_image *src, const mlib_s32 *kernel,
       mlib_s32 m, mlib_s32 n, mlib_s32 dm, mlib_s32 dn,
       mlib_s32 scale, const void *colormap);
```

**Description**
The `mlib_ImageColorErrorDiffusionMxN()` function converts a 3-channel image to a 1-channel indexed image, or converts a 1-channel grayscale image to a 1-channel `MLIB_BIT` image, with the method of error diffusion.

The `src` can be an `MLIB_BYTE` image with 1 or 3 channels. The `dst` must be a 1-channel `MLIB_BIT` or `MLIB_BYTE` image.

The `colormap` must be created by `mlib_ImageColorDitherInit()`. It may or may not have a colorcube included. If it does, the colorcube is used. Otherwise, the general lookup table included in `colormap` is used.

The kernel is required to have the following property:

```
kernel[0] = kernel[1] = ... = kernel[m*dn+dm] = 0;
kernel[m*dn+dm+1] + ... + kernel[m*n-1] = 2**scale;
scale ≥ 0
```

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the error-distribution kernel, in row major order.
- `m` Kernel width. `m > 1`.
- `n` Kernel height. `n > 1`.
- `dm` X coordinate of the key element in the kernel. `0 ≤ dm < m`.
- `dn` Y coordinate of the key element in the kernel. `0 ≤ dn < n`.
- `scale` The scaling factor for kernel to convert the input integer coefficients into floating-point coefficients:

  ```
  floating-point coefficient = integer coefficient * \n  2**(-scale)
  ```
- `colormap` Internal data structure for image dithering.
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageColorDitherInit(3MLIB)`, `mlib_ImageColorDitherFree(3MLIB)`,
- `mlib_ImageColorErrorDiffusion3x3(3MLIB)`,
- `mlib_ImageColorOrderedDither8x8(3MLIB)`,
- `mlib_ImageColorOrderedDitherMxN(3MLIB)`, attributes(5)
The `mlib_ImageColorHSL2RGB()` function performs a conversion from hue/saturation/lightness to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

\[
\begin{align*}
L' &= L \quad \text{if } L \leq 1/2 \\
L' &= 1 - L \quad \text{if } L > 1/2 \\
V &= L + S \times L' \\
P &= L - S \times L' \\
Q &= L + S \times L' \times (1 - 2 \times \text{fraction}(H \times 6)) \\
T &= L - S \times L' \times (1 - 2 \times \text{fraction}(H \times 6)) \\
R, G, B &= V, T, P \quad \text{if } 0 \leq H < 1/6 \\
R, G, B &= Q, V, P \quad \text{if } 1/6 \leq H < 2/6 \\
R, G, B &= P, Q, V \quad \text{if } 2/6 \leq H < 3/6 \\
R, G, B &= P, Q, T \quad \text{if } 3/6 \leq H < 4/6 \\
R, G, B &= T, P, V \quad \text{if } 4/6 \leq H < 5/6 \\
R, G, B &= V, P, Q \quad \text{if } 5/6 \leq H < 1
\end{align*}
\]

where \(0 \leq H < 1\) and \(0 \leq S, L, L', V, P, Q, T, R, G, B \leq 1\).

Assuming a pixel in the source image is \((h, s, l)\) and its corresponding pixel in the destination image is \((r, g, b)\), then for `MLIB_BYTE` images, the following applies:

\[
\begin{align*}
H &= h/256 \\
S &= s/255 \\
L &= l/255 \\
r &= R \times 255 \\
g &= G \times 255 \\
b &= B \times 255
\end{align*}
\]

for `MLIB_SHORT` images, the following applies:

\[
\begin{align*}
H &= (h + 32768)/65536 \\
S &= (s + 32768)/65535 \\
L &= (l + 32768)/65535 \\
r &= R \times 65535 - 32768 \\
g &= G \times 65535 - 32768 \\
b &= B \times 65535 - 32768
\end{align*}
\]

for `MLIB_USHORT` images, the following applies:
\[ H = \frac{h}{65536} \]
\[ S = \frac{s}{65535} \]
\[ L = \frac{l}{65535} \]
\[ r = R \times 65535 \]
\[ g = G \times 65535 \]
\[ b = B \times 65535 \]

and for MLIB_INT images, the following applies:

\[ H = \frac{h + 2147483648}{4294967296} \]
\[ S = \frac{s + 2147483648}{4294967295} \]
\[ L = \frac{l + 2147483648}{4294967295} \]
\[ r = R \times 4294967295 - 2147483648 \]
\[ g = G \times 4294967295 - 2147483648 \]
\[ b = B \times 4294967295 - 2147483648 \]

**Parameters**
The function takes the following arguments:

- \textit{dst} Pointer to destination image.
- \textit{src} Pointer to source image.

**Return Values**
The function returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

**Attributes**
See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
\texttt{mlib\_ImageColorHSL2RGB(3MLIB)}, \texttt{mlib\_ImageColorRGB2HSL(3MLIB)}, \texttt{mlib\_ImageColorRGB2HSL\_Fp(3MLIB)}, \texttt{attributes(5)}
The `mlib_ImageColorHSL2RGB_Fp()` function performs a conversion from hue/saturation/lightness to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

\[
\begin{align*}
L' &= L & \text{if } L \leq 1/2 \\
L' &= 1 - L & \text{if } L > 1/2 \\
V &= L + S*L' \\
P &= L - S*L' \\
Q &= L + S*L'*(1 - 2*\text{fraction}(H*6)) \\
T &= L - S*L'*(1 - 2*\text{fraction}(H*6)) \\
\end{align*}
\]

\[
\begin{align*}
R, G, B &= V, T, P & \text{if } 0 \leq H < 1/6 \\
R, G, B &= Q, V, P & \text{if } 1/6 \leq H < 2/6 \\
R, G, B &= P, V, T & \text{if } 2/6 \leq H < 3/6 \\
R, G, B &= P, Q, V & \text{if } 3/6 \leq H < 4/6 \\
R, G, B &= T, P, V & \text{if } 4/6 \leq H < 5/6 \\
R, G, B &= V, P, Q & \text{if } 5/6 \leq H < 1
\end{align*}
\]

where \(0 \leq H < 1\) and \(0 \leq S, L, L', V, P, Q, T, R, G, B \leq 1\).

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input \(H\) component values must be limited to the \([0.0, 1.0)\) range. Input \(S\) and \(L\) component values must be limited to the \([0.0, 1.0]\) range.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

Multimedia Library Functions - Part 2 217
See Also  
mlib_ImageColorHSL2RGB(3MLIB), mlib_ImageColorRGB2HSL(3MLIB),
mlib_ImageColorRGB2HSL_Fp(3MLIB), attributes(5)
The `mlib_ImageColorHSV2RGB()` function performs a conversion from hue/saturation/value to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

\[
\begin{align*}
P &= V \times (1 - S) \\
Q &= V \times (1 - S \times \text{fraction}(H \times 6)) \\
T &= V \times (1 - S \times (1 - \text{fraction}(H \times 6)))
\end{align*}
\]

\[
\begin{align*}
R, G, B &= V, T, P \quad \text{if } 0 \leq H < 1/6 \\
R, G, B &= Q, V, P \quad \text{if } 1/6 \leq H < 2/6 \\
R, G, B &= P, V, T \quad \text{if } 2/6 \leq H < 3/6 \\
R, G, B &= P, Q, V \quad \text{if } 3/6 \leq H < 4/6 \\
R, G, B &= T, P, V \quad \text{if } 4/6 \leq H < 5/6 \\
R, G, B &= V, P, Q \quad \text{if } 5/6 \leq H < 1
\end{align*}
\]

where \(0 \leq H < 1\) and \(0 \leq S, V, P, Q, T, R, G, B \leq 1\).

Assuming a pixel in the source image is \((h, s, v)\) and its corresponding pixel in the destination image is \((r, g, b)\), then for MLIB_BYTE images, the following applies:

\[
\begin{align*}
H &= h/256 \\
S &= s/255 \\
V &= v/255 \\
r &= R \times 255 \\
g &= G \times 255 \\
b &= B \times 255
\end{align*}
\]

for MLIB_SHORT images, the following applies:

\[
\begin{align*}
H &= (h + 32768)/65536 \\
S &= (s + 32768)/65535 \\
V &= (v + 32768)/65535 \\
r &= R \times 65535 - 32768 \\
g &= G \times 65535 - 32768 \\
b &= B \times 65535 - 32768
\end{align*}
\]

for MLIB_USHORT images, the following applies:

\[
\begin{align*}
H &= h/65536 \\
S &= s/65535 \\
V &= v/65535 \\
r &= R \times 65535 \\
g &= G \times 65535 \\
b &= B \times 65535
\end{align*}
\]
and for MLIB_INT images, the following applies:

\[
H = (h + 2147483648)/4294967296 \\
S = (s + 2147483648)/4294967295 \\
V = (v + 2147483648)/4294967295 \\
r = R*4294967295 - 2147483648 \\
g = G*4294967295 - 2147483648 \\
b = B*4294967295 - 2147483648
\]

Parameters  The function takes the following arguments:

- **dst**  Pointer to destination image.
- **src**  Pointer to source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageColorHSV2RGB(3MLIB), mlib_ImageColorRGB2HSV(3MLIB), mlib_ImageColorRGB2HSV_Fp(3MLIB), attributes(5)
The `mlib_ImageColorHSV2RGB_Fp()` function performs a conversion from hue/saturation/value to red/green/blue. The source and destination images must be three-channel images.

It uses the following equations:

\[
\begin{align*}
P &= V \cdot (1 - S) \\
Q &= V \cdot (1 - S \cdot \text{fraction}(H \cdot 6)) \\
T &= V \cdot (1 - S \cdot (1 - \text{fraction}(H \cdot 6)))
\end{align*}
\]

\[
\begin{align*}
R, G, B &= V, T, P & \text{if } 0 \leq H < 1/6 \\
R, G, B &= Q, V, P & \text{if } 1/6 \leq H < 2/6 \\
R, G, B &= P, V, T & \text{if } 2/6 \leq H < 3/6 \\
R, G, B &= P, Q, V & \text{if } 3/6 \leq H < 4/6 \\
R, G, B &= T, P, V & \text{if } 4/6 \leq H < 5/6 \\
R, G, B &= V, P, Q & \text{if } 5/6 \leq H < 1
\end{align*}
\]

where \(0 \leq H < 1\) and \(0 \leq S, V, P, Q, T, R, G, B \leq 1\).

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input \(H\) component values must be limited to the \([0.0, 1.0)\) range. Input \(S\) and \(V\) component values must be limited to the \([0.0, 1.0]\) range.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

`mlib_ImageColorHSV2RGB(3MLIB)`, `mlib_ImageColorRGB2HSV(3MLIB)`,  
`mlib_ImageColorRGB2HSV_Fp(3MLIB)`, `attributes(5)`
**mlib_ImageColorOrderedDither8x8**

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status (mlib_image *dst, const mlib_image *src, const mlib_s32 *dmask,
    mlib_s32 scale, const void *colormap);
```

**Description**
The `mlib_ImageColorOrderedDither8x8()` function converts a true color image to a pseudo color image with the method of ordered dithering. The source image can be an MLIB_BYTE or MLIB_SHORT image with three or four channels. The destination must be a single-channel MLIB_BYTE or MLIB_SHORT image.

This function works only with a color cube, rather than a general lookup table. The last parameter, `colormap`, is an internal data structure (which may include a color cube) for inverse color mapping. Create it by calling the `mlib_ImageColorTrue2IndexInit()` function.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination or destination image.
- `src` Pointer to source or source image.
- `dmask` Pointer to the 8x8 dither mask, in row major order. The dither mask is transposed differently for different channels to reduce artifacts.
- `scale` Scaling factor for dmask to convert the input integer coefficients into floating-point coefficients:
  ```
  floating-point coefficient = integer coefficient * \n  2**(-scale)
  ```
- `colormap` Internal data structure for inverse color mapping.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageColorErrorDiffusion3x3(3MLIB), mlib_ImageColorTrue2Index(3MLIB), mlib_ImageColorTrue2IndexFree(3MLIB), mlib_ImageColorTrue2IndexInit(3MLIB), attributes(5)`
# mlib_ImageColorOrderedDitherMxN

The `mlib_ImageColorOrderedDitherMxN()` function converts a 3-channel image to a 1-channel indexed image, or converts a 1-channel grayscale image to a 1-channel `MLIB_BIT` image, with the method of ordered dithering.

The `src` can be an `MLIB_BYTE` image with 1 or 3 channels. The `dst` must be a 1-channel `MLIB_BIT` or `MLIB_BYTE` image.

The `colormap` must be created by `mlib_ImageColorDitherInit()`, and it must have a colorcube included.

The dither masks are required to have the following property:

\[ 0 \leq dmask[i][j] < 2^{scale}; \text{ scale } > 0 \]

**Parameters**

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `dmask` Pointer to the dither masks, one per channel, in row major order.
- `m` Mask width. \( m > 1 \).
- `n` Mask height. \( n > 1 \).
- `scale` Scaling factor for `dmask` to convert the input integer coefficients into floating-point coefficients:

\[
\text{floating-point coefficient } = \text{integer coefficient } \times \frac{1}{2^{\text{scale}}} \]

- `colormap` Internal data structure for image dithering.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
mlib_ImageColorOrderedDitherMxN(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageColorDitherInit(3MLIB), mlib_ImageColorDitherFree(3MLIB),  
mlib_ImageColorErrorDiffusion3x3(3MLIB),  
mlib_ImageColorErrorDiffusionMxN(3MLIB),  
mlib_ImageColorOrderedDither8x8(3MLIB), attributes(5)
The `mlib_ImageColorRGB2CIEMono()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:

\[
 dst[x][y][0] = 0.2125*src[x][y][0] + 0.7154*src[x][y][1] + 0.0721*src[x][y][2]
\]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also:

`mlib_ImageColorRGB2CIEMono_fp(3MLIB)`, `mlib_ImageColorRGB2Mono(3MLIB)`, `mlib_ImageColorRGB2Mono_fp(3MLIB)`, `attributes(5)`
mlib_ImageColorRGB2CIEMono_Fp(mlib_image *dst, const mlib_image *src);

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

See Also

mlib_ImageColorRGB2CIEMono(3MLIB), mlib_ImageColorRGB2Mono(3MLIB), mlib_ImageColorRGB2Mono_Fp(3MLIB), attributes(5)
# Name
mlib_ImageColorRGB2HSL - RGB to HSL color conversion

## Synopsis
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorRGB2HSL(mlib_image *dst, const mlib_image *src);
```

## Description
The `mlib_ImageColorRGB2HSL()` function performs a conversion from red/green/blue to hue/saturation/lightness. The source and destination images must be three-channel images.

It uses the following equations:

\[
\begin{align*}
V &= \max(R, G, B) \\
Vmin &= \min(R, G, B) \\
L &= \frac{V + Vmin}{2} \\
S &= \frac{V - Vmin}{V + Vmin} \quad \text{if } L \leq 1/2 \\
S &= \frac{V - Vmin}{2 - V - Vmin} \quad \text{if } L > 1/2 \\
H &= \begin{cases} 
(5.0 + (V - B)/(V - Vmin))/6 & \text{if } R = V \text{ and } G = Vmin \\
(1.0 - (V - G)/(V - Vmin))/6 & \text{if } R = V \text{ and } B = Vmin \\
(1.0 + (V - R)/(V - Vmin))/6 & \text{if } G = V \text{ and } B = Vmin \\
(3.0 - (V - B)/(V - Vmin))/6 & \text{if } G = V \text{ and } R = Vmin \\
(3.0 + (V - G)/(V - Vmin))/6 & \text{if } B = V \text{ and } R = Vmin \\
(5.0 - (V - R)/(V - Vmin))/6 & \text{if } B = V \text{ and } G = Vmin \\
0.0 & \text{if } R = G = B
\end{cases}
\end{align*}
\]

where \(0 \leq R, G, B, V, Vmin, L, S \leq 1\) and \(0 \leq H < 1\).

Assuming a pixel in the source image is \((r, g, b)\) and its corresponding pixel in the destination image is \((h, s, l)\), then for MLIB_BYTE images, the following applies:

\[
\begin{align*}
R &= \frac{r}{255} \\
G &= \frac{g}{255} \\
B &= \frac{b}{255} \\
h &= H*256 \\
s &= S*255 \\
l &= L*255
\end{align*}
\]

for MLIB_SHORT images, the following applies:

\[
\begin{align*}
R &= \frac{(r + 32768)/65535} \\
G &= \frac{(g + 32768)/65535} \\
B &= \frac{(b + 32768)/65535} \\
h &= H*65536 - 32768 \\
s &= S*65535 - 32768 \\
l &= L*65535 - 32768
\end{align*}
\]

for MLIB_USHORT images, the following applies:
\[ R = \frac{r}{65535} \]
\[ G = \frac{g}{65535} \]
\[ B = \frac{b}{65535} \]
\[ h = H \times 65536 \]
\[ s = S \times 65535 \]
\[ l = L \times 65535 \]

and for MLIB_INT images, the following applies:

\[ R = \frac{(r + 2147483648)}{4294967295} \]
\[ G = \frac{(g + 2147483648)}{4294967295} \]
\[ B = \frac{(b + 2147483648)}{4294967295} \]
\[ h = H \times 4294967296 - 2147483648 \]
\[ s = S \times 4294967295 - 2147483648 \]
\[ l = L \times 4294967295 - 2147483648 \]

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes\(\text{(5)}\) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_\_ImageColorHSL\_2\_RGB(3MLIB), mlib_\_ImageColorHSL\_2\_RGB\_Fp(3MLIB),
mlib_\_ImageColorRGB\_2\_HSL\_Fp(3MLIB), attributes(5)
The `mlib_ImageColorRGB2HSL_Fp()` function performs a conversion from red/green/blue to hue/saturation/lightness. The source and destination images must be three-channel images.

It uses the following equations:

\[
V = \max(R, G, B) \\
V_{\text{min}} = \min(R, G, B) \\
L = \frac{V + V_{\text{min}}}{2} \\
S = \begin{cases} 
\frac{V - V_{\text{min}}}{2 - V_{\text{min}}} & \text{if } L \leq 1/2 \\
\frac{V - V_{\text{min}}}{V_{\text{min}}} & \text{if } L > 1/2 
\end{cases} \\
H = \begin{cases} 
\frac{5.0 + (V - B)/(V - V_{\text{min}})}{6} & \text{if } R = V \text{ and } G = V_{\text{min}} \\
\frac{1.0 - (V - G)/(V - V_{\text{min}})}{6} & \text{if } R = V \text{ and } B = V_{\text{min}} \\
\frac{1.0 + (V - R)/(V - V_{\text{min}})}{6} & \text{if } G = V \text{ and } B = V_{\text{min}} \\
\frac{3.0 - (V - B)/(V - V_{\text{min}})}{6} & \text{if } G = V \text{ and } R = V_{\text{min}} \\
\frac{3.0 + (V - G)/(V - V_{\text{min}})}{6} & \text{if } B = V \text{ and } R = V_{\text{min}} \\
\frac{5.0 - (V - R)/(V - V_{\text{min}})}{6} & \text{if } B = V \text{ and } G = V_{\text{min}} \\
0.0 & \text{if } R = G = B
\end{cases}
\]

where \(0 \leq R, G, B, V, V_{\text{min}}, L, S \leq 1\) and \(0 \leq H < 1\).

For `MLIB_FLOAT` and `MLIB_DOUBLE` images, the above equations are followed verbatim. Input \(R, G,\) and \(B\) component values must be limited to the \([0.0, 1.0]\) range.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageColorHSL2RGB(3MLIB), mlib_ImageColorHSL2RGB_Fp(3MLIB), mlib_ImageColorRGB2HSL(3MLIB), attributes(5)
The `mlib_ImageColorRGB2HSV()` function performs a conversion from red/green/blue to hue/saturation/value. The source and destination images must be three-channel images.

It uses the following equations:

\[
\begin{align*}
V &= \text{max}(R, G, B) \\
V_{\text{min}} &= \text{min}(R, G, B) \\
S &= (V - V_{\text{min}})/V \\
H &= \begin{cases} 
(5.0 + (V - B)/(V - V_{\text{min}}))/6 & \text{if } R = V \text{ and } G = V_{\text{min}} \\
(1.0 - (V - G)/(V - V_{\text{min}}))/6 & \text{if } R = V \text{ and } B = V_{\text{min}} \\
(1.0 + (V - R)/(V - V_{\text{min}}))/6 & \text{if } G = V \text{ and } B = V_{\text{min}} \\
(3.0 - (V - B)/(V - V_{\text{min}}))/6 & \text{if } G = V \text{ and } R = V_{\text{min}} \\
(3.0 + (V - G)/(V - V_{\text{min}}))/6 & \text{if } B = V \text{ and } R = V_{\text{min}} \\
(5.0 - (V - R)/(V - V_{\text{min}}))/6 & \text{if } B = V \text{ and } G = V_{\text{min}} \\
0.0 & \text{if } R = G = B
\end{cases}
\end{align*}
\]

where \(0 \leq R, G, B, V, V_{\text{min}}, S \leq 1\) and \(0 \leq H < 1\).

Assuming a pixel in the source image is \((r, g, b)\) and its corresponding pixel in the destination image is \((h, s, v)\), then for MLIB_BYTE images, the following applies:

\[
\begin{align*}
R &= r/255 \\
G &= g/255 \\
B &= b/255 \\
h &= H*256 \\
s &= S*255 \\
v &= V*255
\end{align*}
\]

for MLIB_SHORT images, the following applies:

\[
\begin{align*}
R &= (r + 32768)/65535 \\
G &= (g + 32768)/65535 \\
B &= (b + 32768)/65535 \\
h &= H*65536 - 32768 \\
s &= S*65535 - 32768 \\
v &= V*65535 - 32768
\end{align*}
\]

for MLIB_USHORT images, the following applies:

\[
\begin{align*}
R &= r/65535 \\
G &= g/65535 \\
B &= b/65535
\end{align*}
\]
h = H*65536
s = S*65535
v = V*65535

and for MLIB_INT images, the following applies:

R = (r + 2147483648)/4294967295
G = (g + 2147483648)/4294967295
B = (b + 2147483648)/4294967295
h = H*4294967296 - 2147483648
s = S*4294967295 - 2147483648
v = V*4294967295 - 2147483648

Parameters

The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageColorHSV2RGB(3MLIB), mlib_ImageColorHSV2RGB_Fp(3MLIB),
mlib_ImageColorRGB2HSV_Fp(3MLIB), attributes(5)
The `mlib_ImageColorRGB2HSV_Fp()` function performs a conversion from red/green/blue to hue/saturation/value. The source and destination images must be three-channel images.

It uses the following equations:

\[ V = \max(R, G, B) \]
\[ V_{\text{min}} = \min(R, G, B) \]
\[ S = \frac{V - V_{\text{min}}}{V} \]
\[ H = \begin{cases} 
  \frac{5.0 + (V - B)/(V - V_{\text{min}})}{6} & \text{if } R = V \text{ and } G = V_{\text{min}} \\
  \frac{1.0 - (V - G)/(V - V_{\text{min}})}{6} & \text{if } R = V \text{ and } B = V_{\text{min}} \\
  \frac{1.0 + (V - R)/(V - V_{\text{min}})}{6} & \text{if } G = V \text{ and } B = V_{\text{min}} \\
  \frac{3.0 - (V - B)/(V - V_{\text{min}})}{6} & \text{if } G = V \text{ and } R = V_{\text{min}} \\
  \frac{3.0 + (V - G)/(V - V_{\text{min}})}{6} & \text{if } B = V \text{ and } R = V_{\text{min}} \\
  \frac{5.0 - (V - R)/(V - V_{\text{min}})}{6} & \text{if } B = V \text{ and } G = V_{\text{min}} \\
  0.0 & \text{if } R = G = B 
\end{cases} \]

where \( 0 \leq R, G, B, V, V_{\text{min}}, S \leq 1 \) and \( 0 \leq H < 1 \).

For MLIB_FLOAT and MLIB_DOUBLE images, the above equations are followed verbatim. Input \( R, G, \) and \( B \) component values must be limited to the \([0.0, 1.0]\) range.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
mlib_ImageColorRGB2HSV_Fp(3MLIB)

See Also  
mlib_ImageColorHSV2RGB(3MLIB), mlib_ImageColorHSV2RGB_Fp(3MLIB),  
mlib_ImageColorRGB2HSV(3MLIB), attributes(5)
The `mlib_ImageColorRGB2Mono()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:

\[
\text{dst}[x][y][0] = \text{weight}[0] \times \text{src}[x][y][0] + \\
\text{weight}[1] \times \text{src}[x][y][1] + \\
\text{weight}[2] \times \text{src}[x][y][2]
\]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `weight` Array of three blending coefficients. It is recommended that these sum to 1.0, but it is not required.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageColorRGB2CIEMono(3MLIB)`, `mlib_ImageColorRGB2CIEMono_Fp(3MLIB)`, `mlib_ImageColorRGB2Mono_Fp(3MLIB)`, `attributes(5)`
### mlib_ImageColorRGB2Mono_Fp(3MLIB)

#### Name
mlib_ImageColorRGB2Mono_Fp – RGB to monochrome conversion

#### Synopsis
```
c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorRGB2Mono_Fp(mlib_image *dst,
    const mlib_image *src, const mlib_d64 *weight);
```

#### Description
The `mlib_ImageColorRGB2Mono_Fp()` function performs a conversion from a red/green/blue to a monochromatic image. The source image must be a three-channel image. The destination image must be a single-channel image.

It uses the following equation:
```
dst[x][y][0] = weight[0]*src[x][y][0] +
    weight[1]*src[x][y][1] +
    weight[2]*src[x][y][2]
```

#### Parameters
The function takes the following arguments:
- `dst`  Pointer to destination image.
- `src`  Pointer to source image.
- `weight` Array of three blending coefficients. It is recommended that these sum to 1.0, but it is not required.

#### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

#### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

#### See Also
- `mlib_ImageColorRGB2CIEMono(3MLIB)`, `mlib_ImageColorRGB2CIEMono_Fp(3MLIB)`,
- `mlib_ImageColorRGB2Mono(3MLIB)`, attributes(5)
mlib_ImageColorRGB2XYZ – RGB to XYZ color conversion

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorRGB2XYZ(mlib_image *dst, const mlib_image *src);
```

Description

The `mlib_ImageColorRGB2XYZ()` function performs a color space conversion from ITU-R Rec.708 RGB with D65 white point to CIE XYZ.

The source and destination images must be three-channel images.

It uses the following equation:

```
```

where

```
cmat[] = { 0.412453, 0.357580, 0.180423,  
           0.212671, 0.715160, 0.072169,  
           0.019334, 0.119193, 0.950227 }; 
```

```
src[x][y] = { R, G, B }; 
dst[x][y] = { X, Y, Z }; 
```

Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

See Also

- `mlib_ImageColorConvert1(3MLIB)`, `mlib_ImageColorConvert1_Fp(3MLIB)`, `mlib_ImageColorRGB2XYZ_Fp(3MLIB)`, `mlib_ImageColorXYZ2RGB(3MLIB)`, `mlib_ImageColorYCC2RGB(3MLIB)`, `attributes(5)`
The `mlib_ImageColorRGB2XYZ_Fp()` function performs a color space conversion from ITU-R Rec.708 RGB with D65 white point to CIE XYZ.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{bmatrix}
X \\
Y \\
Z \\
\end{bmatrix} = \begin{bmatrix}
\text{cmat}[0] & \text{cmat}[1] & \text{cmat}[2] \\
\text{cmat}[3] & \text{cmat}[4] & \text{cmat}[5] \\
\text{cmat}[6] & \text{cmat}[7] & \text{cmat}[8] \\
\end{bmatrix} \begin{bmatrix}
R \\
G \\
B \\
\end{bmatrix}
\]

where

\[
\text{cmat}[] = \{ 0.412453, 0.357580, 0.180423, 0.212671, 0.715160, 0.072169, 0.019334, 0.119193, 0.950227 \};
\]

\[
\text{src}[x][y] = \{ R, G, B \} ;
\]

\[
\text{dst}[x][y] = \{ X, Y, Z \} ;
\]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:
**Synopsis**

```
c c [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorRGB2YCC(mlib_image *dst, const mlib_image *src);
```

**Description**

The `mlib_ImageColorRGB2YCC()` function performs a color space conversion from computer RGB' to ITU-R Rec.601 Y’CbCr.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
\end{align*}
\]

where

\[
\begin{align*}
cmat[] & = \{ 65.738/256, 129.057/256, 25.064/256, \\
& -37.945/256, -74.494/256, 112.439/256, \\
& 112.439/256, -94.154/256, -18.285/256 \}; \\
\text{offset[]} & = \{ 16, 128, 128 \}; \\
\text{src}[x][y] & = \{ R', G', B' \}; \\
\text{dst}[x][y] & = \{ Y', Cb, Cr \};
\end{align*}
\]

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageColorConvert2(3MLIB)`, `mlib_ImageColorConvert2_Fp(3MLIB)`,
- `mlib_ImageColorRGB2YCC_Fp(3MLIB)`, `mlib_ImageColorXYZ2RGB(3MLIB)`,
- `mlib_ImageColorXYZ2RGB_Fp(3MLIB)`, `mlib_ImageColorYCC2RGB(3MLIB)`,
- `mlib_ImageColorYCC2RGB_Fp(3MLIB)`
The `mlib_ImageColorRGB2YCC_Fp()` function performs a color space conversion from computer R’G’B’ to ITU-R Rec.601 Y’CbCr.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
|Y'| & = \text{cmat}[0] \cdot |R'| + \text{offset}[0] \\
|Cb| & = \text{cmat}[3] \cdot |G'| + \text{cmat}[4] \cdot |R'| + \text{offset}[1] \\
|Cr| & = \text{cmat}[6] \cdot |B'| + \text{cmat}[7] \cdot |R'| + \text{cmat}[8] \cdot |G'| + \text{offset}[2]
\end{align*}
\]

where

\[
\]

\[
\text{offset}[] = \{ 16, 128, 128 \};
\]

src\[x\][y] = { R’, G’, B’ };

dst\[x\][y] = { Y’, Cb, Cr };

The function takes the following arguments:

dst \quad \text{Pointer to destination image.}

src \quad \text{Pointer to source image.}

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See Also `mlib_ImageColorConvert2(3MLIB)`, `mlib_ImageColorConvert2_Fp(3MLIB)`, `mlib_ImageColorRGB2YCC(3MLIB)`, `mlib_ImageColorXYZ2RGB(3MLIB)`, `mlib_ImageColorXYZ2RGB_Fp(3MLIB)`, `mlib_ImageColorYCC2RGB(3MLIB)`, `mlib_ImageColorYCC2RGB_Fp(3MLIB)`, attributes(5)
The `mlib_ImageColorTrue2Index()` function converts a true color image to a pseudo color image with the method of finding the nearest matched lookup table entry for each pixel. The source image can be an `MLIB_BYTE` or `MLIB_SHORT` image with three or four channels. The destination must be a single-channel `MLIB_BYTE` or `MLIB_SHORT` image.

The last parameter, `colormap`, is an internal data structure (which includes the lookup table) for inverse color mapping. Create it by calling the `mlib_ImageColorTrue2IndexInit()` function.

The function takes the following arguments:

- `dst` Pointer to destination or destination image.
- `src` Pointer to source or source image.
- `colormap` Internal data structure for inverse color mapping.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageColorErrorDiffusion3x3(3MLIB)`, `mlib_ImageColorOrderedDither8x8(3MLIB)`, `mlib_ImageColorTrue2IndexFree(3MLIB)`, `mlib_ImageColorTrue2IndexInit(3MLIB)`, attributes(5)
mlib_ImageColorTrue2IndexFree

**Name**
mlib_ImageColorTrue2IndexFree – releases the internal data structure for true color to indexed color conversion

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_ImageColorTrue2IndexFree(void *colormap);

**Description**
The `mlib_ImageColorTrue2IndexFree()` function releases the internal data structure, `colormap`, which was created by `mlib_ImageColorTrue2IndexInit()` and was used by one of the following functions:

- `mlib_ImageColorTrue2Index`
- `mlib_ImageColorErrorDiffusion3x3`
- `mlib_ImageColorOrderedDither8x8`

**Parameters**
The function takes the following arguments:

- `colormap` Internal data structure for inverse color mapping.

**Return Values**
None.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageColorErrorDiffusion3x3(3MLIB),
mlib_ImageColorOrderedDither8x8(3MLIB),mlib_ImageColorTrue2Index(3MLIB),
mlib_ImageColorTrue2IndexInit(3MLIB),attributes(5)
The `mlib_ImageColorTrue2IndexInit()` function creates and initializes an internal data structure based on the input lookup table and other parameters for inverse color mapping.

The lookup table can have either three or four channels. The number of channels of the lookup table should match that of the source image provided to the function that will use the colormap structure created by this function.

The type of the lookup table can be one of the following:

- **MLIB_BYTE in, MLIB_BYTE out (i.e., BYTE-to-BYTE)**
- **MLIB_SHORT in, MLIB_SHORT out (i.e., SHORT-to-SHORT)**
- **MLIB_SHORT in, MLIB_BYTE out (i.e., SHORT-to-BYTE)**

The input type of the lookup table should match the type of the destination image; the output type of the lookup table should match the source image type. The source and destination images are the images provided to the function that is going to use the colormap structure created by `mlib_ImageColorTrue2IndexInit()` to do inverse color mapping.

The function takes the following arguments:

- **colormap** Internal data structure for inverse color mapping.
- **bits** Number of bits per color component used in the colorcube of the colormap structure. (If `bits = 0`, then no colorcube is created. But the inverse color mapping might be done by using the original lookup table.)
- **intype** Data type of the source image and lookup table.
- **outtype** Data type of the destination indexed image.
- **channels** Number of channels of the lookup table.
- **entries** Number of entries of the lookup table.
- **offset** The first entry offset of the lookup table.
- **table** The lookup table (LUT).

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlibIColorErrorDiffusion3x3(3MLIB),
mlibIColorOrderedDither8x8(3MLIB),mlibIColorTrue2Index(3MLIB),
mlibIColorTrue2IndexFree(3MLIB),attributes(5)
The `mlib_ImageColorXYZ2RGB()` function performs a color space conversion from CIE XYZ to ITU-R Rec.708 RGB with D64 white point.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
    R &= [cmat[0] cmat[1] cmat[2]] \cdot [X] \\
\end{align*}
\]

where

\[
    cmat[] = \{ 3.240479, -1.537150, -0.498535, \\
                 -0.969256, 1.875992, 0.041566, \\
                 0.055648, -0.204043, 1.057311 \};
\]

`src[x][y] = \{ X, Y, Z \};`

`dst[x][y] = \{ R, G, B \};`

Parameters

- The function takes the following arguments:
  - `dst` Pointer to destination image.
  - `src` Pointer to source image.

Return Values

- The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageColorConvert1(3MLIB)`
- `mlib_ImageColorConvert1_Fp(3MLIB)`
- `mlib_ImageColorRGB2XYZ(3MLIB)`
- `mlib_ImageColorRGB2XYZ_Fp(3MLIB)`
- `mlib_ImageColorRGB2YCC(3MLIB)`
- `mlib_ImageColorRGB2YCC_Fp(3MLIB)`
- `mlib_ImageColorXYZ2RGB_Fp(3MLIB)`
- attributes(5)
mlib_ImageColorXYZ2RGB_Fp(3MLIB)

Name  mlib_ImageColorXYZ2RGB_Fp – XYZ to RGB color conversion

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageColorXYZ2RGB_Fp(mlib_image *dst,
const mlib_image *src);

Description  The mlib_ImageColorXYZ2RGB_Fp() function performs a color space conversion from CIE XYZ to ITU-R Rec.709 RGB with D64 white point.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{bmatrix}
|R| \\
|G| \\
|B|
\end{bmatrix}
= \begin{bmatrix}
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]

where

\[
\text{cmat[\]} = \{ 3.240479, -1.537150, -0.498535, \\
-0.969256, 1.875992, 0.041566, \\
0.055648, -0.204043, 1.057311 \};
\]

\[
\text{src[x][y]} = \{ X, Y, Z \};
\text{dst[x][y]} = \{ R, G, B \};
\]

Parameters  The function takes the following arguments:

\[
dst \quad \text{Pointer to destination image.}
\]

\[
src \quad \text{Pointer to source image.}
\]

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageColorConvert1(3MLIB), mlib_ImageColorConvert1_Fp(3MLIB),
mlib_ImageColorRGB2XYZ(3MLIB), mlib_ImageColorRGB2XYZ_Fp(3MLIB),
mlib_ImageColorRGB2YCC(3MLIB), mlib_ImageColorRGB2YCC_Fp(3MLIB),
mlib_ImageColorXYZ2RGB(3MLIB), attributes(5)
The `mlib_ImageColorYCC2RGB()` function performs a color space conversion from ITU-R Rec.601 Y'CbCr to computer R'G'B'.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
|R'| &= \begin{bmatrix} cmat[0] & cmat[1] & cmat[2] \end{bmatrix} |Y'| + |offset[0]| \\
\end{align*}
\]

where

\[
\begin{align*}
cmat[] &= \{ 298.082/256, 0.000/256, 408.583/256, \\
&\quad 298.082/256, -100.291/256, -208.12/256, \\
&\quad 298.082/256, 516.411/256, 0.000/256 \}; \\
offset[] &= \{ -222.922, 135.575, -276.836 \}; \\
src[x][y] &= \{ Y', Cb, Cr \}; \\
dst[x][y] &= \{ R', G', B' \};
\end{align*}
\]

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageColorConvert2(3MLIB)`, `mlib_ImageColorConvert2_Fp(3MLIB)`,
- `mlib_ImageColorRGB2XYZ(3MLIB)`, `mlib_ImageColorRGB2XYZ_Fp(3MLIB)`,
- `mlib_ImageColorRGB2YCC(3MLIB)`, `mlib_ImageColorRGB2YCC_Fp(3MLIB)`,
- `mlib_ImageColorYCC2RGB_Fp(3MLIB)`, `attributes(5)`
The `mlib_ImageColorYCC2RGB_Fp()` function performs a color space conversion from ITU-R Rec.601 Y’CbCr to computer R’G’B’.

The source and destination images must be three-channel images.

It uses the following equation:

\[
\begin{align*}
\end{align*}
\]

where

\[
\begin{align*}
cmat[] &= \{298.082/256, 0.000/256, 408.583/256, \\
&\quad 298.082/256, -100.291/256, -208.120/256, \\
&\quad 298.082/256, 516.411/256, 0.000/256\}; \\
offset[] &= \{ -222.922, 135.575, -276.836 \}; \\
src[x][y] &= \{ Y', Cb, Cr \}; \\
dst[x][y] &= \{ R', G', B' \};
\end{align*}
\]

**Parameters**  The function takes the following arguments:

- `dst`  Pointer to destination image.
- `src`  Pointer to source image.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
- `mlib_ImageColorConvert2(3MLIB)`, `mlib_ImageColorConvert2_Fp(3MLIB)`
- `mlib_ImageColorRGB2XYZ(3MLIB)`, `mlib_ImageColorRGB2XYZ_Fp(3MLIB)`
- `mlib_ImageColorRGB2YCC(3MLIB)`, `mlib_ImageColorRGB2YCC_Fp(3MLIB)`
- `mlib_ImageColorYCC2RGB(3MLIB)`
The `mlib_ImageComposite()` function supports digital image composition. It is a wrapper of the `mlib_ImageBlend_BSCR1_BSRC2` group of functions and can perform various types of composition based on the parameters passed in, whereas each function in that group can perform only the one kind of composition denoted by its name.

The image type must be `MLIB_BYTE`. The input and output images must contain three or four channels. For three-channel images, the alpha value is 1.

The following are predefined blend factor types used in mediaLib image composition functions.

```c
/* image blend factors */
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mlib_blend;
```

See the following table for the definitions of the blend factors.

<table>
<thead>
<tr>
<th>Type</th>
<th>Blend Factor [*]</th>
<th>Abbr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BLEND_ZERO</td>
<td>(0,0,0,0)</td>
<td>ZERO</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE</td>
<td>(1,1,1,1)</td>
<td>ONE</td>
</tr>
<tr>
<td>MLIB_BLEND_DST_COLOR</td>
<td>(Rd,Gd,Bd,Ad)</td>
<td>DC</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_COLOR</td>
<td>(Rs,Gs,Bs,As)</td>
<td>SC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_DST_COLOR</td>
<td>(1,1,1,1)-(Rd,Gd,Bd,Ad)</td>
<td>OMDC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_SRC_COLOR</td>
<td>(1,1,1,1)-(Rs,Gs,Bs,As)</td>
<td>OMSC</td>
</tr>
</tbody>
</table>
Type Blend Factor [*] Abbr.  
MLIB_BLEND_DST_ALPHA (Ad,Ad,Ad,Ad) DA  
MLIB_BLEND_SRC_ALPHA (As,As,As,As) SA  
MLIB_BLEND_ONE_MINUS_DST_ALPHA (1,1,1,1)-(Ad,Ad,Ad,Ad) OMDA  
MLIB_BLEND_ONE_MINUS_SRC_ALPHA (1,1,1,1)-(As,As,As,As) OMSA  
MLIB_BLEND_SRC_ALPHA_SATURATE (f,f,f,1) SAS  

[*]: The components of the first source image pixel are (Rd, Gd, Bd, Ad), and the components of the second source pixel are (Rs, Gs, Bs, As). Function \( f = \min(As, 1-Ad) \).

The blending formula for non-in-place processing is:

\[
Cd = Cs1*S1 + Cs2*S2
\]

where \( Cd \) is the destination pixel \((Rd, Gd, Bd, Ad)\), \( Cs1 \) is the first source pixel \((Rs1, Gs1, Bs1, As1)\), \( Cs2 \) is the second source pixel \((Rs2, Gs2, Bs2, As2)\), and \( S1 \) and \( S2 \) are the blend factors for the first and second sources, respectively.

**Parameters**
The function takes the following arguments:

- \textit{dst} Pointer to destination image.
- \textit{src1} Pointer to the first source image.
- \textit{src2} Pointer to the second source image.
- \textit{bsrc1} Blend factor type for the first source image.
- \textit{bsrc2} Blend factor type for the second source image.
- \textit{cmask} Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. \( cmask \) must be either 0x01 or 0x08.

**Return Values**
The function returns \textit{MLIB_SUCCESS} if successful. Otherwise it returns \textit{MLIB_FAILURE}.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_ImageBlend_BSRC1_BSRC2(3MLIB), mlib_ImageBlend_BSRC1_BSRC2_Inp(3MLIB),  
mlib_ImageComposite_Inp(3MLIB), attributes(5)
The mlib_ImageComposite_Inp() function supports digital image composition. It is a wrapper of the mlib_ImageBlend_BSCR1_BSRC2_Inp group of functions and can perform various types of composition based on the parameters passed in, whereas each function in the mlib_ImageBlend_BSCR1_BSRC2_Inp group can perform only the one kind of composition denoted by its name.

The image type must be MLIB_BYTE. The input and output images must contain three or four channels. For three-channel images, the alpha value is as if the alpha value is 1.

The following are predefined blend factor types used in mediaLib image composition functions.

```c
typedef enum {
    MLIB_BLEND_ZERO,
    MLIB_BLEND_ONE,
    MLIB_BLEND_DST_COLOR,
    MLIB_BLEND_SRC_COLOR,
    MLIB_BLEND_ONE_MINUS_DST_COLOR,
    MLIB_BLEND_ONE_MINUS_SRC_COLOR,
    MLIB_BLEND_DST_ALPHA,
    MLIB_BLEND_SRC_ALPHA,
    MLIB_BLEND_ONE_MINUS_DST_ALPHA,
    MLIB_BLEND_ONE_MINUS_SRC_ALPHA,
    MLIB_BLEND_SRC_ALPHA_SATURATE
} mlib_blend;
```

See the following table for the definitions of the blend factors.

<table>
<thead>
<tr>
<th>Type</th>
<th>Blend Factor [*]</th>
<th>Abbr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BLEND_ZERO</td>
<td>(0,0,0,0)</td>
<td>ZERO</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE</td>
<td>(1,1,1,1)</td>
<td>ONE</td>
</tr>
<tr>
<td>MLIB_BLEND_DST_COLOR</td>
<td>(Rd,Gd,Bd,Ad)</td>
<td>DC</td>
</tr>
<tr>
<td>MLIB_BLEND_SRC_COLOR</td>
<td>(Rs,Gs, Bs, As)</td>
<td>SC</td>
</tr>
<tr>
<td>MLIB_BLEND_ONE_MINUS_DST_COLOR</td>
<td>(1,1,1,1)-(Rd,Gd,Bd,Ad)</td>
<td>OMDC</td>
</tr>
</tbody>
</table>
Type Blend Factor [*] Abbr.
MLIB_BLEND_ONE_MINUS_SRC_COLOR (1,1,1,1)-(Rs,Gs,Bs,As) OMSC
MLIB_BLEND_DST_ALPHA (Ad,Ad,Ad,Ad) DA
MLIB_BLEND_SRC_ALPHA (As,As,As,As) SA
MLIB_BLEND_ONE_MINUS_DST_ALPHA (1,1,1,1)-(Ad,Ad,Ad,Ad) OMDA
MLIB_BLEND_ONE_MINUS_SRC_ALPHA (1,1,1,1)-(As,As,As,As) OMSA
MLIB_BLEND_SRC_ALPHA_SATURATE (f,f,f,1) SAS

[*]: The components of the first source image pixel are (Rd, Gd, Bd, Ad), and the components of the second source pixel are (Rs, Gs, Bs, As). Function \( f = \min(As, 1-Ad) \). The first source image is also the destination image.

The blending formula for in-place processing is:

\[ Cd = Cd*D + Cs*S \]

where \( Cd \) is the destination pixel (Rd, Gd, Bd, Ad), \( Cs \) is the source pixel (Rs, Gs, Bs, As), and \( D \) and \( S \) are the blend factors for the destination and source, respectively.

**Parameters**
The function takes the following arguments:
- \( src1dst \) Pointer to the first source and the destination image.
- \( src2 \) Pointer to the second source image.
- \( bsrc1 \) Blend factor type for the first source image.
- \( bsrc2 \) Blend factor type for the second source image.
- \( cmask \) Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit is the alpha channel. cmask must be either 0x01 or 0x08.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:
See Also  
mlib_ImageBlend_BSRC1_BSRC2(3MLIB), mlib_ImageBlend_BSRC1_BSRC2_Inp(3MLIB),  
mlib_ImageComposite(3MLIB), attributes(5)
mlib_ImageConstAdd – addition with a constant

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConstAdd(mlib_image *dst, const mlib_image *src,
const mlib_s32 *c);

Description

The mlib_ImageConstAdd() function adds a constant to an image on a pixel-by-pixel basis.

It uses the following equation:

dst[x][y][i] = c[i] + src[x][y][i]

Parameters

The function takes the following arguments:

dst  Pointer to destination image.
src  Pointer to source image.
c    An array of constants to be added to each pixel by channel. c[i] contains the constant for channel i.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageConstAdd_Fp(3MLIB), mlib_ImageConstAdd_Fp_Inp(3MLIB), mlib_ImageConstAdd_Inp(3MLIB), attributes(5)
The `mlib_ImageConstAdd_Fp()` function adds a constant to a floating-point image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = c[i] + \text{src}[x][y][i]
\]

The function takes the following arguments:

- **dst**  
  Pointer to destination image.
- **src**  
  Pointer to source image.
- **c**  
  An array of constants to be added to each pixel by channel. \(c[i]\) contains the constant for channel \(i\).

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageConstAdd(3MLIB)`, `mlib_ImageConstAdd_Fp_Inp(3MLIB)`
- `mlib_ImageConstAdd_Inp(3MLIB)`, `attributes(5)`
The `mlib_ImageConstAdd_Fp_Inp()` function adds a constant to a floating-point image on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = c[i] + \text{srcdst}[x][y][i]
\]

**Parameters**

- `srcdst`  
  Pointer to source and destination image.
- `c`  
  An array of constants to be added to each pixel by channel. `c[i]` contains the constant for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageConstAdd(3MLIB), mlib_ImageConstAdd_Fp(3MLIB), mlib_ImageConstAdd_Inp(3MLIB), attributes(5)`
mlib_ImageConstAdd_Inp

Name mlib_ImageConstAdd_Inp – addition with a constant

Synopsis cc { flag... } file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConstAdd_Inp(mlib_image *srcdst, const mlib_s32 *c);

Description The mlib_ImageConstAdd_Inp() function adds a constant to an image on a pixel-by-pixel basis, in place.

It uses the following equation:

```
srcdst[x][y][i] = c[i] + srcdst[x][y][i]
```

Parameters The function takes the following arguments:

- **srcdst**   Pointer to source and destination image.
- **c**   An array of constants to be added to each pixel by channel. c[i] contains the constant for channel i.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageConstAdd(3MLIB), mlib_ImageConstAdd_Fp(3MLIB), mlib_ImageConstAdd_Fp_Inp(3MLIB), attributes(5)
# Name
mlib_ImageConstAnd – And with a constant

# Synopsis
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConstAnd(mlib_image *dst, const mlib_image *src,
                          const mlib_s32 *c);
```

# Description
The mlib_ImageConstAnd() function computes the And of the source image with a constant.

It uses the following equation:

\[
\text{dst}[x][y][i] = c[i] \& \text{src}[x][y][i]
\]

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

# Parameters
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

# Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

# Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

# See Also
mlib_ImageConstAnd_Inp(3MLIB), attributes(5)
# mlib_ImageConstAnd_Inp

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageConstAnd_Inp(mlib_image *srcdst, const mlib_s32 *c);
```

## Description

The `mlib_ImageConstAnd_Inp()` function computes the And of the source image with a constant, in place.

It uses the following equation:

$$
srcdst[x][y][i] = c[i] \& srcdst[x][y][i]
$$

The data type of the images can be `MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT`.

## Parameters

- **srcdst**: Pointer to first source and destination image.
- **c**: Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`mlib_ImageConstAnd(3MLIB), attributes(5)`
The `mlib_ImageConstAndNot()` function computes the And of the Not of the source image with a constant.

It uses the following equation:

\[
dst[x][y][i] = c[i] \& (~\text{src}[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageConstAndNot_Inp(3MLIB)`, attributes(5)
The `mlib_ImageConstAndNot_Inp()` function computes the And of the Not of the source image with a constant, in place.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \text{c}[i] \& (\neg \text{srcdst}[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

### Parameters

- `srcdst` Pointer to first source and destination image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageConstAndNot(3MLIB)`, `attributes(5)`
The `mlib_ImageConstDiv()` function divides each pixel in an image into a constant value on a pixel-by-pixel basis.

It uses the following equation:

\[
dst[x][y][i] = c[i] / src[x][y][i]
\]

In the case of \( src[x][y][i] = 0 \),

\[
dst[x][y][i] = 0 \quad \text{if} \ c[i] = 0 \\
dst[x][y][i] = \text{DATA\_TYPE\_MAX} \quad \text{if} \ c[i] > 0 \\
dst[x][y][i] = \text{DATA\_TYPE\_MIN} \quad \text{if} \ c[i] < 0
\]

where DATA\_TYPE is MLIB\_U8, MLIB\_S16, MLIB\_U16, or MLIB\_S32 for an image of type MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT, or MLIB\_INT, respectively.

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **c** Constant into which each pixel is divided. \( c[i] \) contains the constant for channel \( i \).

The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

See also `mlib_ImageConstDiv_Fp(3MLIB)`, `mlib_ImageConstDiv_Fp_Inp(3MLIB)`, `mlib_ImageConstDiv_Inp(3MLIB)`, `mlib_ImageConstDivShift(3MLIB)`, `mlib_ImageConstDivShift_Inp(3MLIB)`, attributes(5)
The `mlib_ImageConstDiv_Fp()` function divides each pixel in a floating-point image by a constant value on a pixel-by-pixel basis. It uses the following equation:

\[
\text{dst}[x][y][i] = \frac{c[i]}{\text{src}[x][y][i]}
\]

where the operation follows the IEEE-754 standard.

**Parameters**
- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Constant into which each pixel is divided. `c[i]` contains the constant for channel `i`.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageConstDiv(3MLIB)`, `mlib_ImageConstDiv_Fp_Inp(3MLIB)`,
- `mlib_ImageConstDiv_Inp(3MLIB)`, `mlib_ImageConstDivShift(3MLIB)`,
- `mlib_ImageConstDivShift_Inp(3MLIB)`, attributes(5)
mlib_ImageConstDiv_Fp_Inp(3MLIB)

**Name**  
mlib_ImageConstDiv_Fp_Inp – division into a constant

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  
mlib_status mlib_ImageConstDiv_Fp_Inp(mlib_image *srcdst,  
const mlib_d64 *c);

**Description**  
The `mlib_ImageConstDiv_Fp_Inp()` function divides each pixel in a floating-point image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = \frac{c[i]}{\text{srcdst}[x][y][i]} \]

where the operation follows the IEEE-754 standard.

**Parameters**  
The function takes the following arguments:

- **srcdst**  
  Pointer to source and destination image.

- **c**  
  Constant into which each pixel is divided. c[i] contains the constant for channel i.

**Return Values**  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageConstDiv(3MLIB), mlib_ImageConstDiv_Fp(3MLIB),  
mlib_ImageConstDiv_Inp(3MLIB), mlib_ImageConstDivShift(3MLIB),  
mlib_ImageConstDivShift_Inp(3MLIB), attributes(5)
The `mlib_ImageConstDiv_Inp()` function divides each pixel in an image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = c[i] / \text{srcdst}[x][y][i]
\]

In the case of \(\text{srcdst}[x][y][i] = 0\),

\[
\begin{align*}
\text{srcdst}[x][y][i] &= 0 & \text{if } c[i] = 0 \\
\text{srcdst}[x][y][i] &= \text{DATA TYPE}_\text{MAX} & \text{if } c[i] > 0 \\
\text{srcdst}[x][y][i] &= \text{DATA TYPE}_\text{MIN} & \text{if } c[i] < 0
\end{align*}
\]

where \(\text{DATA TYPE}\) is \(\text{MLIB}_\text{U8}\), \(\text{MLIB}_\text{S16}\), \(\text{MLIB}_\text{U16}\), or \(\text{MLIB}_\text{S32}\) for an image of type \(\text{MLIB}_\text{BYTE}\), \(\text{MLIB}_\text{SHORT}\), \(\text{MLIB}_\text{USHORT}\), or \(\text{MLIB}_\text{INT}\), respectively.

### Parameters
The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `c` Constant into which each pixel is divided. \(c[i]\) contains the constant for channel \(i\).

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
`mlib_ImageConstDiv(3MLIB)`, `mlib_ImageConstDiv_Fp(3MLIB)`, `mlib_ImageConstDiv_Fp_Inp(3MLIB)`, `mlib_ImageConstDivShift(3MLIB)`, `mlib_ImageConstDivShift_Inp(3MLIB)`, attributes(5)
The `mlib_ImageConstDivShift()` function divides each pixel in an image into a constant value on a pixel-by-p-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

$$\text{dst}[x][y][i] = \frac{\text{c}[i]}{\text{src}[x][y][i]} \times 2^{\text{shift}}$$

In the case of $\text{src}[x][y][i]=0$,

- $\text{dst}[x][y][i] = 0$ if $\text{c}[i] = 0$
- $\text{dst}[x][y][i] = \text{DATA\_TYPE\_MAX}$ if $\text{c}[i] > 0$
- $\text{dst}[x][y][i] = \text{DATA\_TYPE\_MIN}$ if $\text{c}[i] < 0$

where \text{DATA\_TYPE} is \text{MLIB\_U8}, \text{MLIB\_S16}, \text{MLIB\_U16}, or \text{MLIB\_S32} for an image of type \text{MLIB\_BYTE}, \text{MLIB\_SHORT}, \text{MLIB\_USHORT}, or \text{MLIB\_INT}, respectively.

**Parameters**
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **c** Constant into which each pixel is divided. $\text{c}[i]$ contains the constant for channel $i$.
- **shift** Left shifting factor. $0 \leq \text{shift} \leq 31$.

**Return Values**
The function returns \text{MLIB\_SUCCESS} if successful. Otherwise it returns \text{MLIB\_FAILURE}.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageConstDiv(3MLIB), mlib_ImageConstDiv_Fp(3MLIB), mlib_ImageConstDiv_Fp_Inp(3MLIB), mlib_ImageConstDiv_Inp(3MLIB), mlib_ImageConstDivShift_Inp(3MLIB), attributes(5)`
**Name**  
mlib_ImageConstDivShift_Inp – division into a constant, with shifting

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_ImageConstDivShift_Inp(mlib_image *srcdst,
const mlib_s32 *c, mlib_s32 shift);
```

**Description**  
The `mlib_ImageConstDivShift_Inp()` function divides each pixel in an image into a constant value on a pixel-by-pixel basis, in place. It scales the result by a left shift and writes the result to the image on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = c[i] / \text{srcdst}[x][y][i] \times 2^{\text{shift}} \]

In the case of \( \text{srcdst}[x][y][i] = 0 \),
- \( \text{srcdst}[x][y][i] = 0 \) if \( c[i] = 0 \)
- \( \text{srcdst}[x][y][i] = \text{DATA\_TYPE\_MAX} \) if \( c[i] > 0 \)
- \( \text{srcdst}[x][y][i] = \text{DATA\_TYPE\_MIN} \) if \( c[i] < 0 \)

where \( \text{DATA\_TYPE} \) is \( \text{MLIB\_U8}, \text{MLIB\_S16}, \text{MLIB\_U16}, \) or \( \text{MLIB\_S32} \) for an image of type \( \text{MLIB\_BYTE}, \text{MLIB\_SHORT}, \text{MLIB\_USHORT}, \) or \( \text{MLIB\_INT} \), respectively.

**Parameters**  
The function takes the following arguments:
- `srcdst`  
  Pointer to source and destination image.
- `c`  
  Constant into which each pixel is divided. \( c[i] \) contains the constant for channel \( i \).
- `shift`  
  Left shifting factor. \( 0 \leq \text{shift} \leq 31 \).

**Return Values**  
The function returns \( \text{MLIB\_SUCCESS} \) if successful. Otherwise it returns \( \text{MLIB\_FAILURE} \).

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageConstDiv(3MLIB), mlib_ImageConstDiv_Fp(3MLIB),  
mlib_ImageConstDiv_Fp_Inp(3MLIB), mlib_ImageConstDiv_Inp(3MLIB),  
mlib_ImageConstDivShift(3MLIB), attributes(5)
The `mlib_ImageConstMul()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][i] = c[i] \times \text{src}[x][y][i] \]

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Constant by which each pixel is multiplied. `c[i]` contains the constant for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageConstMul_Fp(3MLIB), mlib_ImageConstMul_Fp_Inp(3MLIB), mlib_ImageConstMul_Inp(3MLIB), mlib_ImageConstMulShift(3MLIB), mlib_ImageConstMulShift_Inp(3MLIB), attributes(5)`
The `mlib_ImageConstMul_Fp()` function multiplies each pixel in a floating-point image by a constant value on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][i] = c[i] \times \text{src}[x][y][i] \]

### Parameters
The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **c**: Constant by which each pixel is multiplied. \( c[i] \) contains the constant for channel \( i \).

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageConstMul(3MLIB)`
- `mlib_ImageConstMul_Fp_Inp(3MLIB)`
- `mlib_ImageConstMul_Inp(3MLIB)`
- `mlib_ImageConstMulShift(3MLIB)`
- `mlib_ImageConstMulShift_Inp(3MLIB)`, attributes(5)
The \texttt{mlib\_ImageConstMul\_Fp\_Inp} function multiplies each pixel in a floating-point image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = c[i] \times \text{srcdst}[x][y][i] \]

The function takes the following arguments:

- \textit{srcdst} Pointer to source and destination image.
- \textit{c} Constant by which each pixel is multiplied. \( c[i] \) contains the constant for channel \( i \).

The function returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

See also \texttt{mlib\_ImageConstMul(3MLIB)}, \texttt{mlib\_ImageConstMul\_Fp(3MLIB)}, \texttt{mlib\_ImageConstMul\_Inp(3MLIB)}, \texttt{mlib\_ImageConstMul\_Shift(3MLIB)}, \texttt{mlib\_ImageConstMul\_Shift\_Inp(3MLIB)}, \texttt{attributes(5)}
The `mlib_ImageConstMul_Inp()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = c[i] \times \text{srcdst}[x][y][i]
\]

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `c` Constant by which each pixel is multiplied. \(c[i]\) contains the constant for channel \(i\).

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageConstMul(3MLIB)`, `mlib_ImageConstMul_Fp(3MLIB)`,
- `mlib_ImageConstMul_Fp_Inp(3MLIB)`, `mlib_ImageConstMulShift(3MLIB)`,
- `mlib_ImageConstMulShift_Inp(3MLIB)`, attributes(5)
The `mlib_ImageConstMulShift()` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = c[i] \times \text{src}[x][y][i] \times 2^{(-\text{shift})}
\]

**Parameters**

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **c** Constant by which each pixel is multiplied. \(c[i]\) contains the constant for channel \(i\).
- **shift** Right shifting factor. 0 ≤ shift ≤ 31.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageConstMul(3MLIB)`
- `mlib_ImageConstMul_Fp(3MLIB)`
- `mlib_ImageConstMul_Fp_Inp(3MLIB)`
- `mlib_ImageConstMul_Inp(3MLIB)`
- `mlib_ImageConstMulShift_Inp(3MLIB)`, attributes(5)
The `mlib_ImageConstMulShift_Inp` function multiplies each pixel in an image by a constant value on a pixel-by-pixel basis. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = c[i] \times \text{srcdst}[x][y][i] \times 2^{-\text{shift}} \]

**Parameters** The function takes the following arguments:

- **srcdst** Pointer to source and destination image.
- **c** Constant by which each pixel is multiplied. \( c[i] \) contains the constant for channel \( i \).
- **shift** Right shifting factor. \( 0 \leq \text{shift} \leq 31 \).

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageConstMul(3MLIB)`, `mlib_ImageConstMul_Fp(3MLIB)`,
- `mlib_ImageConstMul_Fp_Inp(3MLIB)`, `mlib_ImageConstMul_Inp(3MLIB)`,
- `mlib_ImageConstMulShift(3MLIB)`, `attributes(5)`
# mlib_ImageConstNotAnd

**mlib_ImageConstNotAnd()** function computes the logical And of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

\[
    \text{dst}[x][y][i] = \neg(c[i] \& \text{src}[x][y][i])
\]

The data type of the images can be **MLIB_BIT**, **MLIB_BYTE**, **MLIB_SHORT**, **MLIB_USHORT**, or **MLIB_INT**.

### Parameters
- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **c** Array of constants to be applied to each pixel. c[i] contains the constant for channel i.

### Return Values
The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- mlib_ImageConstNotAnd_Inp(3MLIB), attributes(5)
The `mlib_ImageConstNotAnd_Inp()` function computes the logical And of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = \neg(c[i] \& \text{srcdst}[x][y][i]) \]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:

- `srcdst` Pointer to first source and destination image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageConstNotAnd(3MLIB)` and `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
The `mlib_ImageConstNotOr()` function computes the logical Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

\[
dst[x][y][i] = \neg(c[i] \mid src[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

### Parameters
- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **c** Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageConstNotOr_Inp(3MLIB)`, `attributes(5)`

---

```
include <mlib.h>

mlib_status mlib_ImageConstNotOr(mlib_image *dst, const mlib_image *src, const mlib_s32 *c);
```
The `mlib_ImageConstNotOr_Inp()` function computes the logical Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

```
srcdst[x][y][i] = ~(c[i] | srcdst[x][y][i])
```

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

### Parameters

- `srcdst` Pointer to first source and destination image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageConstNotOr_Inp(3MLIB)`, attributes(5)
mlib_ImageConstNotXor(3MLIB)

**Name**  
mlib_ImageConstNotXor – NotXor with a constant

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConstNotXor(mlib_image *dst, const mlib_image *src,  
      const mlib_s32 *c);

**Description**  
The `mlib_ImageConstNotXor()` function computes the logical exclusive Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][i] = \neg(c[i] \ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ ^{\ }
The `mlib_ImageConstNotXor_Inp()` function computes the logical exclusive Or of the source image with a constant and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
srcest[x][y][i] = \neg (c[i] \lor srcest[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
- `srcest` Pointer to first source and destination image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageConstNotXor(3MLIB), attributes(5)`
mlib_ImageConstOr() function computes the logical Or of the source image with a constant on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][i] = c[i] \mid \text{src}[x][y][i] \]

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

The function takes the following arguments:
- \( dst \) Pointer to destination image.
- \( src \) Pointer to source image.
- \( c \) Array of constants to be applied to each pixel. \( c[i] \) contains the constant for channel \( i \).

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_ImageConstOr_Inp(3MLIB), attributes(5)
The `mlib_ImageConstOr_Inp()` function computes the logical Or of the source image with a constant on a pixel-by-p-pixel basis, in place.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \text{c}[i] \lor \text{srcdst}[x][y][i]
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:

- `srcdst` Pointer to first source and destination image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageConstOr(3MLIB), attributes(5)`
# mlib_ImageConstOrNot

## Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
```

```
#include <mlib.h>

mlib_status mlib_ImageConstOrNot(mlib_image *dst, const mlib_image *src,
                                 const mlib_s32 *c);
```

## Description

The `mlib_ImageConstOrNot()` function computes the logical Not of the source image and then takes the logical Or of that result with a constant on a pixel-by-pixel basis.

It uses the following equation:

```
dst[x][y][i] = c[i] | (~src[x][y][i])
```

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

## Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`mlib_ImageConstOrNot_Inp(3MLIB)`,
The `mlib_ImageConstOrNot_Inp()` function computes the logical Not of the source image and then takes the logical Or of that result with a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
srcdst[x][y][i] = c[i] | (-srcdst[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

### Parameters

The function takes the following arguments:

- `srcdst`: Pointer to first source and destination image.
- `c`: Array of constants to be applied to each pixel. \(c[i]\) contains the constant for channel \(i\).

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageConstOrNot(3MLIB)`, `attributes(5)`
The `mlib_ImageConstSub()` function subtracts an image pixel from a constant on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][i] = c[i] - \text{src}[x][y][i] \]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Array of constants from which each pixel is subtracted by channel. \( c[i] \) contains the constant for channel \( i \).

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageConstSub_Fp(3MLIB), mlib_ImageConstSub_Fp_Inp(3MLIB), mlib_ImageConstSub_Inp(3MLIB), attributes(5)`
mlib_ImageConstSub_Fp – Subtraction with a constant

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConstSub_Fp(mlib_image *dst, const mlib_image *src,
const mlib_d64 *c);

Description

The mlib_ImageConstSub_Fp() function subtracts a floating-point image pixel from a
constant on a pixel-by-pixel basis.

It uses the following equation:

dst[x][y][i] = c[i] - src[x][y][i]

Parameters

The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
c Array of constants from which each pixel is subtracted by channel. c[i] contains the
constant for channel i.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageConstSub(3MLIB), mlib_ImageConstSub_Fp(3MLIB),
mlib_ImageConstSub_Inp(3MLIB), attributes(5)
The `mlib_ImageConstSub_Fp_Inp()` function subtracts a floating-point image pixel from a constant on a pixel-by-pixel basis, in place. It uses the following equation:

\[ \text{srcdst}[x][y][i] = c[i] - \text{srcdst}[x][y][i] \]

The function takes the following arguments:

- `srcdst`  Pointer to source and destination image.
- `c`  Array of constants from which each pixel is subtracted by channel. `c[i]` contains the constant for channel `i`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageConstSub(3MLIB), mlib_ImageConstSub_Fp(3MLIB), mlib_ImageConstSub_Inp(3MLIB), attributes(5)`
The `mlib_ImageConstSub_Inp()` function subtracts an image pixel from a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

```
srcdst[x][y][i] = c[i] - srcdst[x][y][i]
```

**Parameters**
- `srcdst` Pointer to source and destination image.
- `c` Array of constants from which each pixel is subtracted by channel. `c[i]` contains the constant for channel `i`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageConstSub(3MLIB), mlib_ImageConstSub_Fp(3MLIB), mlib_ImageConstSub_Fp_Inp(3MLIB), attributes(5)`
## mlib_ImageConstXor

### Synopsis

```c
#include <mlib.h>
mlib_status mlib_ImageConstXor(mlib_image *dst, const mlib_image *src,
                                const mlib_s32 *c);
```

### Description

The `mlib_ImageConstXor()` function computes the exclusive Or of the source image with a constant on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = c[i] \oplus \text{src}[x][y][i]
\]

The data type of the images can be `MLIB.Bit`, `MLIB.Byte`, `MLIB.Short`, `MLIB.UsShort`, or `MLIB.Int`.

### Parameters

- **`dst`** Pointer to destination image.
- **`src`** Pointer to source image.
- **`c`** Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

### Return Values

The function returns `MLIB.Success` if successful. Otherwise it returns `MLIB.Failure`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageConstXor_Inp(3MLIB)`, attributes(5)
The `mlib_ImageConstXor_Inp()` function computes the exclusive Or of the source image with a constant on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
    \text{srcdst}[x][y][i] = c[i] \oplus \text{srcdst}[x][y][i]
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:

- `srcdst` Pointer to first source and destination image.
- `c` Array of constants to be applied to each pixel. `c[i]` contains the constant for channel `i`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

See also `mlib_ImageConstXor(3MLIB), attributes(5)`
The `mlib_ImageConv2x2()` function performs a 2x2 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at \((0, 0)\) of the kernel matrix.

It uses the following equation:

\[
\begin{align*}
    dst[x][y][i] &= \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{-scale} \\
    p &= -dm, \quad q &= -dn
\end{align*}
\]

where \(m = 2\), \(n = 2\), \(dm = (m - 1)/2 = 0\), \(dn = (n - 1)/2 = 0\).

**Parameters**

The function takes the following arguments:

- `dst`  
  Pointer to destination image.

- `src`  
  Pointer to source image.

- `kernel`  
  Pointer to the convolution kernel, in row major order.

- `scale`  
  Scaling factor.

- `cmask`  
  Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

- `edge`  
  Type of edge condition. It can be one of the following:

- `MLIB_EDGE_DST_NO_WRITE`
- `MLIB_EDGE_DST_FILL_ZERO`
- `MLIB_EDGE_DST_COPY_SRC`
- `MLIB_EDGE_SRC_EXTEND`
Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageConv2x2_Fp(3MLIB), mlib_ImageConv2x2Index(3MLIB),
mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB),
mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB),
mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB),
mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),
mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),
mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
# mlib_ImageConv2x2_Fp

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageConv2x2_Fp(mlib_image *dst, const mlib_image *src,
                                 const mlib_d64 *kernel, mlib_s32 cmask, mlib_edge edge);
```

## Description

The `mlib_ImageConv2x2_Fp()` function performs a 2x2 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (0, 0) of the kernel matrix.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i]*k[p][q]
\]

where \( m = 2, n = 2, dm = (m - 1)/2 = 0, dn = (n - 1)/2 = 0 \).

## Parameters

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **kernel**: Pointer to the convolution kernel, in row major order.
- **cmask**: Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:
mlib_ImageConv2x2_Fp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB), mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv2x2Index()` function performs a 2x2 convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at \((0, 0)\) of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single-channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] \times k[p][q] \times 2^{-scale}
\]

where \(m = 2\), \(n = 2\), \(dm = (m - 1)/2 = 0\), \(dn = (n - 1)/2 = 0\).

### Parameters
The function takes the following arguments:

- **\(dst\)** Pointer to destination image.
- **\(src\)** Pointer to source image.
- **\(kernel\)** Pointer to the convolution kernel, in row major order.
- **\(scale\)** Scaling factor.
- **\(edge\)** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
- **\(colormap\)** Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
          mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB),
          mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB),
          mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB),
          mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),
          mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),
          mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
          mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
          mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
          mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
          mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
          mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
          mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
          mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv3x3()` function performs a 3x3 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

The data type of source and destination images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i]\times2^{-\text{scale}} \times k[p][q]\times2^i
\]

where \( m = 3 \), \( n = 3 \), \( dm = (m - 1)/2 = 1 \), \( dn = (n - 1)/2 = 1 \).

**Parameters**

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the convolution kernel, in row major order.
- `scale` Scaling factor.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_imageConv2x2(3MLIB), mlib_imageConv2x2_fp(3MLIB),
mlib_imageConv2x2Index(3MLIB), mlib_imageConv3x3_fp(3MLIB),
mlib_imageConv3x3Index(3MLIB), mlib_imageConv4x4(3MLIB),
mlib_imageConv4x4_fp(3MLIB), mlib_imageConv4x4Index(3MLIB),
mlib_imageConv5x5(3MLIB), mlib_imageConv5x5_fp(3MLIB),
mlib_imageConv5x5Index(3MLIB), mlib_imageConv7x7(3MLIB),
mlib_imageConv7x7_fp(3MLIB), mlib_imageConv7x7Index(3MLIB),
mlib_imageConvKernelConvert(3MLIB), mlib_imageConvMxN(3MLIB),
mlib_imageConvMxN_fp(3MLIB), mlib_imageConvMxNIndex(3MLIB),
mlib_imageConvolveMxN(3MLIB), mlib_imageConvolveMxN_fp(3MLIB),
mlib_imageSConv3x3(3MLIB), mlib_imageSConv3x3_fp(3MLIB),
mlib_imageSConv5x5(3MLIB), mlib_imageSConv5x5_fp(3MLIB),
mlib_imageSConv7x7(3MLIB), mlib_imageSConv7x7_fp(3MLIB),
mlib_imageSConvKernelConvert(3MLIB), attributes(5)
mlib_ImageConv3x3_Fp

**Name**  
mlib_ImageConv3x3_Fp - 3x3 convolution

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_ImageConv3x3_Fp(mlib_image *dst, const mlib_image *src,  
const mlib_d64 *kernel, mlib_s32 cmask, mlib_edge edge);
```

**Description**  
The `mlib_ImageConv3x3_Fp()` function performs a floating-point 3x3 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1} \sum_{q=-dn}^{n-1} src[x+p][y+q][i] * k[p][q]
\]

where  
\[m = 3, \quad n = 3, \quad dm = (m - 1)/2 = 1, \quad dn = (n - 1)/2 = 1.\]

**Parameters**  
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

- **kernel**  
  Pointer to the convolution kernel, in row major order.

- **cmask**  
  Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.

- **edge**  
  Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:
mlib_ImageConv3x3_Fp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB),
mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB),
mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),
mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),
mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The **mlib_ImageConv3x3Index**() function performs a 3x3 convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single-channel images. The image type must be **MLIB_BYTE** or **MLIB_SHORT**.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{dm} \sum_{q=-dn}^{dn} \text{src}[x+p][y+q][i] * \text{kernel}[p][q] * 2^{(-\text{scale})}
\]

where \( m = 3 \), \( n = 3 \), \( dm = (m - 1)/2 = 1 \), \( dn = (n - 1)/2 = 1 \).

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **kernel**: Pointer to the convolution kernel, in row major order.
- **scale**: Scaling factor.
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND
- **colormap**: Internal data structure for inverse color mapping. This data structure is generated by the **mlib_ImageColorTrue2IndexInit()** function. The source and destination images must be single-channel images.
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
          mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
          mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv4x4(3MLIB),
          mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB),
          mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),
          mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),
          mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
          mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
          mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
          mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
          mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
          mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
          mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv4x4()` function performs a 4x4 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at $(1, 1)$ of the kernel matrix.

It uses the following equation:

$$
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{-scale}$$

where $m = 4$, $n = 4$, $dm = (m - 1)/2 = 1$, $dn = (n - 1)/2 = 1$.

**Parameters**

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **kernel** Pointer to the convolution kernel, in row major order.
- **scale** Scaling factor.
- **cmask** Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge** Type of edge condition. It can be one of the following:

  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
          mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3Index(3MLIB),
          mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4Index(3MLIB),
          mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),
          mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),
          mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
          mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
          mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
          mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
          mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
          mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
          mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
          mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv4x4_Fp()` function performs a floating-point 4x4 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at (1, 1) of the kernel matrix.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-\text{dm}}^{\text{dm}} \sum_{q=-\text{dn}}^{\text{dn}} \text{src}[x+p][y+q][i]*\text{kernel}[p][q]
\]

where \( m = 4 \), \( n = 4 \), \( \text{dm} = (m - 1)/2 = 1 \), \( \text{dn} = (n - 1)/2 = 1 \).

**Parameters**

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the convolution kernel, in row major order.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:
mlib_ImageConv4x4_Fp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
- mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
- mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
- mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4Index(3MLIB),
- mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),
- mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),
- mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
- mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
- mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
- mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
- mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
- mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
- mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
- mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv4x4Index()` function performs a 4x4 convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at (1, 1) of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] \times k[p][q] \times 2^{(-\text{scale})}
\]

where \( m = 4 \), \( n = 4 \), \( dm = (m - 1)/2 = 1 \), \( dn = (n - 1)/2 = 1 \).

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the convolution kernel, in row major order.
- `scale` Scaling factor.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
- `colormap` Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB), mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB), mlib_Imlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB), mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv5x5()` function performs a 5x5 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i]*k[p][q]*2^{(-scale)}
\]

where \( m = 5 \), \( n = 5 \), \( dm = (m - 1)/2 = 2 \), \( dn = (n - 1)/2 = 2 \).

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the convolution kernel, in row major order.
- `scale` Scaling factor.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:

  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND
Return Values: The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB), mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB), mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv5x5_Fp()` function performs a floating-point 5x5 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{dm-1} \sum_{q=-dn}^{dn-1} \text{src}[x+p][y+q][i] \times k[p][q]
\]

where \( m = 5 \), \( n = 5 \), \( dm = (m - 1)/2 = 2 \), \( dn = (n - 1)/2 = 2 \).

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the convolution kernel, in row major order.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:
mlib_imageConv5x5(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_imageConv2x2(3MLIB), mlib_imageConv2x2_Fp(3MLIB),  
mlib_imageConv2x2Index(3MLIB), mlib_imageConv3x3(3MLIB),  
mlib_imageConv3x3_Fp(3MLIB), mlib_imageConv3x3Index(3MLIB),  
mlib_imageConv4x4(3MLIB), mlib_imageConv4x4_Fp(3MLIB),  
mlib_imageConv4x4Index(3MLIB), mlib_imageConv5x5(3MLIB),  
mlib_imageConv5x5Index(3MLIB), mlib_imageConv7x7(3MLIB),  
mlib_imageConv7x7_Fp(3MLIB), mlib_imageConv7x7Index(3MLIB),  
mlib_imageConvKernelConvert(3MLIB), mlib_imageConvMxN(3MLIB),  
mlib_imageConvMxN_Fp(3MLIB), mlib_imageConvMxNIndex(3MLIB),  
mlib_imageConvolveMxN(3MLIB), mlib_imageConvolveMxN_Fp(3MLIB),  
mlib_imageSConv3x3(3MLIB), mlib_imageSConv3x3_Fp(3MLIB),  
mlib_imageSConv5x5(3MLIB), mlib_imageSConv5x5_Fp(3MLIB),  
mlib_imageSConv7x7(3MLIB), mlib_imageSConv7x7_Fp(3MLIB),  
mlib_imageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv5x5Index()` function performs a 5x5 convolution on a color indexed source image by using the user-supplied kernel. The source and destination images must be single-channel images.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i]*k[p][q]*2^{(-scale)}
\]

where \( m = 5 \), \( n = 5 \), \( dm = (m - 1)/2 = 2 \), \( dn = (n - 1)/2 = 2 \).

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **kernel**: Pointer to the convolution kernel, in row major order.
- **scale**: Scaling factor.
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
- **colormap**: Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv7x7(3MLIB),
mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxMxN(3MLIB),
mlib_ImageConvMxMxN_Fp(3MLIB), mlib_ImageConvolveMxMxN(3MLIB),
mlib_ImageConvolveMxMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB),
mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB),
mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB),
mlib_ImageSConv7x7_Fp(3MLIB), mlib_ImageConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv7x7()` function performs a 7x7 convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{dm} \sum_{q=-dn}^{dn} \text{src}[x+p][y+q][i] \times k[p][q] \times 2^{\text{scale}}
\]

where \( m = 7 \), \( n = 7 \), \( dm = (m - 1)/2 = 3 \), \( dn = (n - 1)/2 = 3 \).

### Parameters
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `kernel` Pointer to the convolution kernel, in row major order.
- `scale` Scaling factor.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),
mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
mlib_ImageConv7x7_Fp

Name
mlib_ImageConv7x7_Fp – 7x7 convolution

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConv7x7_Fp(mlib_image *dst, const mlib_image *src,
                                 const mlib_d64 *kernel, mlib_s32 cmask, mlib_edge edge);

Description
The mlib_ImageConv7x7_Fp() function performs a floating-point 7x7 convolution on the
source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same
number of channels. The unselected channels in the output image are not overwritten. For
single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at the center of the
kernel matrix.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] \cdot k[p][q]
\]

where \( m = 7 \), \( n = 7 \), \( dm = (m - 1)/2 = 3 \), \( dn = (n - 1)/2 = 3 \).

Parameters
The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
kernel Pointer to the convolution kernel, in row major order.
cmask Channel mask to indicate the channels to be convolved, each bit of which
        represents a channel in the image. The channels corresponding to bits with a value
        of 1 are those to be processed. For a single-channel image, the channel mask is
        ignored.
edge Type of edge condition. It can be one of the following:
        MLIB_EDGE_DST_NO_WRITE
        MLIB_EDGE_DST_FILL_ZERO
        MLIB_EDGE_DST_COPY_SRC
        MLIB_EDGE_SRC_EXTEND

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:
### mlib_ImageConv7x7_Fp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),  
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),  
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),  
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),  
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),  
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),  
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7Index(3MLIB),  
mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),  
mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),  
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),  
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),  
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),  
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),  
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConv7x7Index()` function performs a 7x7 convolution on the color indexed source image by using the user-supplied kernel. The source and destination images must be single-channel images.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at the center of the kernel matrix.

This function performs the convolution on color indexed image. The input image and the output image must be single channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] \times k[p][q] \times 2^{-\text{scale}}
\]

where \( m = 7 \), \( n = 7 \), \( dm = (m - 1)/2 = 3 \), \( dn = (n - 1)/2 = 3 \).

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **kernel**: Pointer to the convolution kernel, in row major order.
- **scale**: Scaling factor.
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
- **colormap**: Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.
Return Values  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  `mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),
mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)`
The `mlib_ImageConvKernelConvert()` function converts a floating-point convolution kernel to an integer kernel with its scaling factor suitable to be used in convolution functions.

The function takes the following arguments:

- `ikernel` Pointer to integer convolution kernel, in row major order.
- `iscale` Pointer to scaling factor of the integer convolution kernel.
- `fkernel` Pointer to floating-point convolution kernel, in row major order.
- `m` Width of the convolution kernel. \( m \geq 1 \).
- `n` Height of the convolution kernel. \( n \geq 1 \).
- `type` The image type. It can be one of the following:
  - MLIB_BIT
  - MLIB_BYTE
  - MLIB_SHORT
  - MLIB_USHORT
  - MLIB_INT

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See [attributes(5)](../attributes(5)) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageConv2x2(3MLIB)`, `mlib_ImageConv2x2_Fp(3MLIB)`
- `mlib_ImageConv3x3(3MLIB)`, `mlib_ImageConv3x3Index(3MLIB)`
- `mlib_ImageConv4x4(3MLIB)`, `mlib_ImageConv4x4_Fp(3MLIB)`
- `mlib_ImageConv5x5(3MLIB)`, `mlib_ImageConv5x5Index(3MLIB)`
- `mlib_ImageConv7x7(3MLIB)`, `mlib_ImageConv7x7_Fp(3MLIB)`
- `mlib_ImageConvolveMxN(3MLIB)`, `mlib_ImageConvolveMxN_Fp(3MLIB)`
mlib_ImageConvKernelConvert(3MLIB)

mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConvMxN()` function performs a MxN convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at $(dm, dn)$ of the kernel matrix.

It uses the following equation:

$$dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] \times k[p][q] \times 2^{-scale}$$

where $m \geq 1$, $n \geq 1$, $0 \leq dm < m$, $0 \leq dn < n$.

### Parameters

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **kernel** Pointer to the convolution kernel, in row major order.
- **m** Width of the convolution kernel. $m \geq 1$.
- **n** Height of the convolution kernel. $n \geq 1$.
- **dm** X coordinate of the key element in the convolution kernel. $0 \leq dm < m$.
- **dn** Y coordinate of the key element in the convolution kernel. $0 \leq dn < n$.
- **scale** Scaling factor.
- **cmask** Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge** Type of edge condition. It can be one of the following:
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_imageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConvMxN_Fp()` function performs a MxN convolution on the source image by using the user-supplied kernel.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

For this convolution, the key element of the convolution kernel is located at \((dm, dn)\) of the kernel matrix.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i]*k[p][q]
\]

where \(m \geq 1\), \(n \geq 1\), \(0 \leq dm < m\), \(0 \leq dn < n\).

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **kernel**: Pointer to the convolution kernel, in row major order.
- **m**: Width of the convolution kernel. \(m \geq 1\).
- **n**: Height of the convolution kernel. \(n \geq 1\).
- **dm**: X coordinate of the key element in the convolution kernel. \(0 \leq dm < m\).
- **dn**: Y coordinate of the key element in the convolution kernel. \(0 \leq dn < n\).
- **cmask**: Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to bits with a value of 1 are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxNIndex(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageConvolveMxN(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB),
mlib_ImageConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConvMxNIndex()` function performs a MxN convolution on the color indexed source image by using the user-supplied kernel.

The input and output images must have the same image type and size.

For this convolution, the key element of the convolution kernel is located at \((dm, dn)\) of the kernel matrix.

This function performs the convolution on a color indexed image. The input image and the output image must be single-channel images. The image type must be `MLIB_BYTE` or `MLIB_SHORT`.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] * k[p][q] * 2^{-scale}
\]

where \(m > 1\), \(n > 1\), \(0 \leq dm < m\), \(0 \leq dn < n\).

### Parameters

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **kernel** Pointer to the convolution kernel, in row major order.
- **m** Width of the convolution kernel. \(m > 1\).
- **n** Height of the convolution kernel. \(n > 1\).
- **dm** X coordinate of the key element in the convolution kernel. \(0 \leq dm < m\).
- **dn** Y coordinate of the key element in the convolution kernel. \(0 \leq dn < n\).
- **scale** Scaling factor.
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageConv2x2(3MLIB),mlib_ImageConv2x2_Fp(3MLIB), mlib_ImageConv2x2Index(3MLIB),mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB),mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB),mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB),mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB),mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB),mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB),mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB),mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvolveMxN(3MLIB),mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB),mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB),mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB),mlib_ImageSConv7x7_Fp(3MLIB), mlib_ImageSConvKernelConvert(3MLIB),attributes(5)
mlib_ImageConvolveMxN

mlib_ImageConvolveMxN — MxN convolution, with kernel analysis for taking advantage of special cases

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageConvolveMxN(mlib_image *dst, const mlib_image *src,
const mlib_d64 *kernel, mlib_s32 m, mlib_s32 n, mlib_s32 dm,
mlib_s32 dn, mlib_s32 cmask, mlib_edge edge);

Description

The mlib_ImageConvolveMxN() function analyzes the convolution kernel, converts the floating-point kernel to an integer kernel, then performs a MxN convolution on the source image by calling either one of the functions like mlib_ImageSConv3x3(), mlib_ImageConv3x3(), and etc. in special cases or mlib_ImageConvMxN() in other cases.

The input image and the output image must have the same image type and have the same number of channels. The unslected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

It uses the following equation:

\[ dst[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} src[x+p][y+q][i] \times k[p][q] \]

where \( m \geq 1, n \geq 1, 0 \leq dm < m, 0 \leq dn < n \).

Parameters

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **kernel** Pointer to the convolution kernel, in row major order.
- **m** Width of the convolution kernel. \( m \geq 1 \).
- **n** Height of the convolution kernel. \( n \geq 1 \).
- **dm** X coordinate of the key element in the convolution kernel. \( 0 \leq dm < m \).
- **dn** Y coordinate of the key element in the convolution kernel. \( 0 \leq dn < n \).
- **cmask** Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge** Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB), mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB), mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB), mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB), mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB), mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB), mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB), mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB), mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB), mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB), mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageConvolveMxN_Fp()` function analyzes the convolution kernel, then performs a MxN convolution on the source image by calling either one of the functions like `mlib_ImageSConv3x3_Fp()`, `mlib_ImageConv3x3_Fp()`, and etc. in special cases or `mlib_ImageConvMxN_Fp()` in other cases.

The input image and the output image must have the same image type and have the same number of channels. The unselected channels in the output image are not overwritten. For single-channel images, the channel mask is ignored.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-\text{dm}}^{\text{m}-1-\text{dm}} \sum_{q=-\text{dn}}^{\text{n}-1-\text{dn}} \text{src}[x+p][y+q][i] \times k[p][q]
\]

where \(m \geq 1\), \(n \geq 1\), \(0 \leq \text{dm} < m\), \(0 \leq \text{dn} < n\).

### Parameters

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **kernel** Pointer to the convolution kernel, in row major order.
- **m** Width of the convolution kernel. \(m \geq 1\).
- **n** Height of the convolution kernel. \(n \geq 1\).
- **dm** X coordinate of the key element in the convolution kernel. \(0 \leq \text{dm} < m\).
- **dn** Y coordinate of the key element in the convolution kernel. \(0 \leq \text{dn} < n\).
- **cmask** Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB),
mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB),
mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
# mlib_ImageCopy

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageCopy(mlib_image *dst, const mlib_image *src);
```

## Description

The `mlib_ImageCopy()` function copies the source image to the destination image. The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

It uses the following equation:

\[ \text{dst}[x][y][i] = \text{src}[x][y][i] \]

## Parameters

- **`dst`**: Pointer to destination image.
- **`src`**: Pointer to source image.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_ImageCopyArea(3MLIB)`, `mlib_ImageCopyMask(3MLIB)`, `mlib_ImageCopyMask_Fp(3MLIB)`, `mlib_ImageCopySubimage(3MLIB)`
# mlib_ImageCopyArea

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageCopyArea(mlib_image *img, mlib_s32 x, mlib_s32 y,
                               mlib_s32 w, mlib_s32 h, mlib_s32 dx, mlib_s32 dy);
```

## Description

The `mlib_ImageCopyArea()` function copies a specified rectangular area from one portion of the image to another portion of the same image. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

It uses the following equation:

\[
\text{img}[x+dx+i][y+dy+j][i] = \text{img}[x+i][y+j][i]
\]

where \( i = 0, 1, \ldots, \text{w}-1 \); \( j = 0, 1, \ldots, \text{h}-1 \).

## Parameters

The function takes the following arguments:

- **img**: Pointer to source image.
- **x**: X coordinate of the area origin in the source.
- **y**: Y coordinate of the area origin in the source.
- **w**: Width of the area to be copied.
- **h**: Height of the area to be copied.
- **dx**: Horizontal displacement in pixels of the area to be copied.
- **dy**: Vertical displacement in pixels of the area to be copied.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`mlib_ImageCopy(3MLIB)`, `mlib_ImageCopyMask(3MLIB)`, `mlib_ImageCopyMask_Fp(3MLIB)`, `mlib_ImageCopySubimage(3MLIB)`, `attributes(5)`
The `mlib_ImageCopyMask` function copies one image to another image via a mask image by using it as a yes/no indicator. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][i] &= \text{src}[x][y][i] \quad \text{if mask}[x][y][i] \leq \text{thresh}[i] \\
\text{dst}[x][y][i] &= \text{dst}[x][y][i] \quad \text{if mask}[x][y][i] > \text{thresh}[i]
\end{align*}
\]

### Parameters
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `mask` Pointer to mask image.
- `thresh` Threshold for the mask image. `thresh[i]` contains the threshold for channel `i`.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: `mlib_ImageCopy(3MLIB)`, `mlib_ImageCopyArea(3MLIB)`, `mlib_ImageCopyMask_Fp(3MLIB)`, `mlib_ImageCopySubimage(3MLIB)`, attributes(5)
The `mlib_ImageCopyMask_Fp()` function copies one image to another image via a mask image by using it as a yes/no indicator. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][i] &= \text{src}[x][y][i] \quad \text{if} \quad \text{mask}[x][y][i] \leq \text{thresh}[i] \\
\text{dst}[x][y][i] &= \text{dst}[x][y][i] \quad \text{if} \quad \text{mask}[x][y][i] > \text{thresh}[i]
\end{align*}
\]

**Parameters**

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `mask` Pointer to mask image.
- `thresh` Threshold for the mask image. `thresh[i]` contains the threshold for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageCopy(3MLIB)`
- `mlib_ImageCopyArea(3MLIB)`
- `mlib_ImageCopyMask(3MLIB)`
- `mlib_ImageCopySubimage(3MLIB)`
- attributes(5)
mlib_ImageCopySubimage(3MLIB)

Name  mlib_ImageCopySubimage – copy subimage

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageCopySubimage(mlib_image *dst, const mlib_image *src,
    mlib_s32 xd, mlib_s32 yd, mlib_s32 xs, mlib_s32 ys, mlib_s32 w,
    mlib_s32 h);

Description  The mlib_ImageCopySubimage() function copies a specified rectangular area from one image
to a specified area of another image. The data type of the images can be MLIB_BIT, MLIB_BYTE,
MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

It uses the following equation:

\[ \text{dst}[\text{xd}+i][\text{yd}+j][i] = \text{src}[\text{xs}+i][\text{ys}+j][i] \]

where \( i = 0, 1, \ldots, w-1 \); \( j = 0, 1, \ldots, h-1 \).

Parameters  The function takes the following arguments:

\[ \begin{align*}
    \text{dst} & \quad \text{Pointer to destination image.} \\
    \text{src} & \quad \text{Pointer to source image.} \\
    \text{xd} & \quad \text{X coordinate of the area origin in the destination.} \\
    \text{yd} & \quad \text{Y coordinate of the area origin in the destination.} \\
    \text{xs} & \quad \text{X coordinate of the area origin in the source.} \\
    \text{ys} & \quad \text{Y coordinate of the area origin in the source.} \\
    \text{w} & \quad \text{Width of the area to be copied.} \\
    \text{h} & \quad \text{Height of the area to be copied.}
\end{align*} \]

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

\[
\begin{array}{|c|c|}
\hline
\text{ATTRIBUTE TYPE} & \text{ATTRIBUTE VALUE} \\
\hline
\text{Interface Stability} & \text{Committed} \\
\text{MT-Level} & \text{MT-Safe} \\
\hline
\end{array}
\]

See Also  mlib_ImageCopy(3MLIB), mlib_ImageCopyArea(3MLIB), mlib_ImageCopyMask(3MLIB), mlib_ImageCopyMask_Fp(3MLIB), attributes(5)
mlib_ImageCreate – image creation

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_image *mlib_ImageCreate(mlib_type type, mlib_s32 channels,
                             mlib_s32 width, mlib_s32 height);
```

**Description**

The `mlib_ImageCreate()` function creates a mediaLib image data structure and allocates memory space for image data. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

To obtain the best performance, it is recommended that you use this function to create a mediaLib image whenever possible, as this guarantees alignment.

**Parameters**

The function takes the following arguments:

- `type` : Image data type.
- `channels` : Number of channels in the image.
- `width` : Width of image in pixels.
- `height` : Height of image in pixels.

**Return Values**

The function returns a pointer to the `mlib_image` data structure.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageCreateStruct(3MLIB), mlib_ImageCreateSubimage(3MLIB), mlib_ImageDelete(3MLIB), mlib_ImageSetPaddings(3MLIB), attributes(5)`
The `mlib_ImageCreateStruct()` function creates a mediaLib image data structure with parameters supplied by the user.

**Parameters**

The function takes the following arguments:

- `type` Image data type. It can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.
- `channels` Number of channels in the image.
- `width` Width of image in pixels.
- `height` Height of image in pixels.
- `stride` Stride of each row of the data space in bytes.
- `datbuf` Pointer to the image data buffer.

**Return Values**

The function returns a pointer to the `mlib_image` data structure.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageCreate(3MLIB)`, `mlib_ImageCreateSubimage(3MLIB)`,
- `mlib_ImageSetStruct(3MLIB)`, `mlib_ImageResetStruct(3MLIB)`,
- `mlib_ImageDelete(3MLIB)`, `mlib_ImageSetFormat(3MLIB)`,
- `mlib_ImageSetPaddings(3MLIB)`, `attributes(5)`
mlib_ImageCreateSubimage — subimage creation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_image *mlib_ImageCreateSubimage(mlib_image *img, mlib_s32 x,
mlib_s32 y, mlib_s32 w, mlib_s32 h);

Description

The mlib_ImageCreateSubimage() function creates a mediaLib image data structure for a
subimage based on a source image. Note that the memory space of the source image data is
used for the subimage data. The data type of the image can be MLIB_BIT, MLIB_BYTE,
MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters

The function takes the following arguments:

- **img** Pointer to source image.
- **x** X coordinate of subimage origin in the source.
- **y** Y coordinate of subimage origin in the source.
- **w** Width of the subimage in pixels.
- **h** Height of the subimage in pixels.

Return Values

The function returns a pointer to the mlib_image data structure.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageCreate(3MLIB), mlib_ImageCreateStruct(3MLIB),
mlib_ImageDelete(3MLIB), mlib_ImageSetPaddings(3MLIB), attributes(5)
**Synopsis**

```
cc [ flag... ] file... -lmlib [ library...]
#include <mlib.h>

mlib_status mlib_ImageCrossCorrel(mlib_d64 *correl, const mlib_image *img1,
                          const mlib_image *img2);
```

**Description**

The `mlib_ImageCrossCorrel()` function computes the cross-correlation between a pair of images.

It uses the following equation:

\[
    \text{correl}[i] = \frac{1}{w \cdot h} \times \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] \times \text{img2}[x][y][i])
\]

where \( w \) and \( h \) are the width and height of the images, respectively.

**Parameters**

The function takes the following arguments:

- `correl` Pointer to cross correlation array on a channel basis. The array must be the size of channels in the images. `correl[i]` contains the cross-correlation of channel \( i \).
- `img1` Pointer to first image.
- `img2` Pointer to second image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageAutoCorrel(3MLIB), mlib_ImageAutoCorrel_Fp(3MLIB),
- `mlib_ImageCrossCorrel_Fp(3MLIB), mlib_ImageNormCrossCorrel(3MLIB),
- `mlib_ImageNormCrossCorrel_Fp(3MLIB), attributes(5)"
The `mlib_ImageCrossCorrel_Fp()` function computes the cross-correlation between a pair of floating-point images.

It uses the following equation:

\[
\text{correl}[i] = \frac{1}{w \times h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] \times \text{img2}[x][y][i])
\]

where \( w \) and \( h \) are the width and height of the images, respectively.

**Parameters**
- `correl` Pointer to cross correlation array on a channel basis. The array must be the size of channels in the images. `correl[i]` contains the cross-correlation of channel \( i \).
- `img1` Pointer to first image.
- `img2` Pointer to second image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageAutoCorrel(3MLIB)`, `mlib_ImageAutoCorrel_Fp(3MLIB)`, `mlib_ImageCrossCorrel(3MLIB)`, `mlib_ImageNormCrossCorrel(3MLIB)`, `mlib_ImageNormCrossCorrel_Fp(3MLIB)`, `attributes(5)`
The `mlib_ImageDataTypeConvert()` function converts between data types `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, and `MLIB_DOUBLE`.

The input and output data images must have the same width, height, and number of channels. Conversion to a smaller pixel format clamps the source value to the dynamic range of the destination pixel.

See the following table for available variations of the data type conversion function.

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Dest. Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_BYTE</code></td>
<td><code>MLIB_BIT</code></td>
<td><code>(x &gt; 0) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_SHORT</code></td>
<td><code>MLIB_BIT</code></td>
<td><code>(x &gt; 0) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_USHORT</code></td>
<td><code>MLIB_BIT</code></td>
<td><code>(x &gt; 0) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_INT</code></td>
<td><code>MLIB_BIT</code></td>
<td><code>(x &gt; 0) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_FLOAT</code></td>
<td><code>MLIB_BIT</code></td>
<td><code>(x &gt; 0) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_DOUBLE</code></td>
<td><code>MLIB_BIT</code></td>
<td><code>(x &gt; 0) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_BIT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(x == 1) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_SHORT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)clamp(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_USHORT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)clamp(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_INT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)clamp(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_FLOAT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)clamp(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_DOUBLE</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)clamp(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_BIT</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(x == 1) ? 1 : 0</code></td>
</tr>
<tr>
<td><code>MLIB_BYTE</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(mlib_s16)x</code></td>
</tr>
<tr>
<td><code>MLIB_USHORT</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(mlib_s16)clamp(x, -32768, 32767)</code></td>
</tr>
<tr>
<td><code>MLIB_INT</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(mlib_s16)clamp(x, -32768, 32767)</code></td>
</tr>
<tr>
<td><code>MLIB_FLOAT</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(mlib_s16)clamp(x, -32768, 32767)</code></td>
</tr>
<tr>
<td>Source Type</td>
<td>Dest. Type</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_SHORT</td>
<td>(mlib_s16)clamp(x, -32768, 32767)</td>
</tr>
<tr>
<td>MLIB_BIT</td>
<td>MLIB_USHORT</td>
<td>(x == 1)?1:0</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_BIT</td>
<td>MLIB_INT</td>
<td>(x == 1)?1:0</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_INT</td>
<td>(mlib_s32)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_INT</td>
<td>(mlib_s32)x</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>MLIB_INT</td>
<td>(mlib_s32)x</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_INT</td>
<td>(mlib_s32)clamp(x, -2147483647/-1, 2147483647)</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_INT</td>
<td>(mlib_s32)clamp(x, -2147483647/-1, 2147483647)</td>
</tr>
<tr>
<td>MLIB_BIT</td>
<td>MLIB_FLOAT</td>
<td>(x == 1)?1.0:0.0</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_BIT</td>
<td>MLIB_DOUBLE</td>
<td>(x == 1)?1.0:0.0</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
</tbody>
</table>

Multimedia Library Functions - Part 2

345
The actions are defined in C-style pseudo-code. All type casts follow the rules of standard C.
clamp() can be defined as a macro:

```
#define clamp(x, low, high) (((x) < (low)) ? (low) : (((x) > (high)) ? (high) : (x)))
```

**Parameters**
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageDataTypeConvert(3MLIB), attributes(5)
# mlib_ImageDelete (3MLIB)

**Name**  
mlib_ImageDelete – image delete

**Synopsis**  
```c
cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

void mlib_ImageDelete(mlib_image *img);
```

**Description**  
The `mlib_ImageDelete()` function deletes the mediaLib image data structure and frees the memory space of the image data only if it is allocated through `mlib_ImageCreate()`. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

**Parameters**  
The function takes the following arguments:

- `img`: Pointer to mediaLib image structure.

**Return Values**  
None.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_ImageCreate(3MLIB), mlib_ImageCreateStruct(3MLIB), mlib_ImageCreateSubimage(3MLIB), attributes(5)`
The `mlib_ImageDilate4()` function performs a dilation operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 4-neighbor binary images, it uses the following equation:

\[
dst[x][y][0] = \text{OR}\{ \text{src}[x][y][0], \text{src}[x-1][y][0], \text{src}[x+1][y][0], \text{src}[x][y-1][0], \text{src}[x][y+1][0] \}
\]

For 4-neighbor grayscale images, it uses the following equation:

\[
dst[x][y][0] = \text{MAX}\{ \text{src}[x][y][0], \text{src}[x-1][y][0], \text{src}[x+1][y][0], \text{src}[x][y-1][0], \text{src}[x][y+1][0] \}
\]

where \( x = 1, \ldots, w - 2; \ y = 1, \ldots, h - 2 \).

### Parameters

The function takes the following arguments:

- **dst** Pointer to destination image
- **src** Pointer to source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageDilate4_Fp(3MLIB)`, `mlib_ImageDilate8(3MLIB)`, `mlib_ImageDilate8_Fp(3MLIB)`, attributes(5)
The `mlib_ImageDilate4_Fp()` function performs a floating-point dilation operation on an image by using each pixel’s four orthogonal neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 4-neighbor grayscale images, it uses the following equation:

\[
dst[x][y][0] = \max\{ src[x][y][0], \\
src[x-1][y][0], src[x+1][y][0], \\
src[x][y-1][0], src[x][y+1][0] \}
\]

where \( x = 1, \ldots, w-2; \ y = 1, \ldots, h-2. \)

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageDilate4(3MLIB), mlib_ImageDilate8(3MLIB), mlib_ImageDilate8_Fp(3MLIB), attributes(5)`
mlib_ImageDilate8() performs a dilation operation on an image by using all eight of each pixel’s neighbors. The source and destination images must be single-channel images. The data type can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

For 8-neighbor binary images, it uses the following equation:

\[
dst[x][y][0] = \text{OR} \{ \text{src}[p][q][0], x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}
\]

For 8-neighbor grayscale images, it uses the following equation:

\[
dst[x][y][0] = \text{MAX} \{ \text{src}[p][q][0], x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}
\]

where \( x = 1, \ldots, w-2; \ y = 1, \ldots, h-2. \)

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Parameters

- \( dst \) Point to destination image.
- \( src \) Point to source image.

See Also

mlib_ImageDilate4(3MLIB), mlib_ImageDilate4_Fp(3MLIB), mlib_ImageDilate8_Fp(3MLIB), attributes(5)
The `mlib_ImageDilate8_Fp()` function performs a floating-point dilation operation on an image by using all eight of each pixel's neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 8-neighbor grayscale images, it uses the following equation:

\[
\text{dst}[x][y][0] = \max\{ \text{src}[p][q][0], \quad \begin{array}{l}
    x-1 \leq p \leq x+1; \quad y-1 \leq q \leq y+1
\end{array}\}
\]

where \(x = 1, \ldots, w-2; \quad y = 1, \ldots, h-2\).

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageDilate4(3MLIB), mlib_ImageDilate4_Fp(3MLIB), mlib_ImageDilate8(3MLIB), attributes(5)`
The `mlib_ImageDiv1_Fp_Inp()` function divides the second floating-point source image into the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

\[ \text{src1dst}[x][y][i] = \frac{\text{src1dst}[x][y][i]}{\text{src2}[x][y][i]} \]

where the operation follows the IEEE-754 standard.

**Parameters**

- `src1dst`: Pointer to first source and destination image.
- `src2`: Pointer to second source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageDiv_Fp(3MLIB)`
- `mlib_ImageDiv2_Fp_Inp(3MLIB)`
- `attributes(5)`
The `mlib_ImageDiv2_Fp_Inp()` function divides the second floating-point source image into the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

\[ \text{src2dst}[x][y][i] = \frac{\text{src1}[x][y][i]}{\text{src2dst}[x][y][i]} \]

where the operation follows the IEEE-754 standard.

### Parameters
- **src2dst** Pointer to second source and destination image.
- **src1** Pointer to first source image.

### Return Values
The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageDiv_Fp(3MLIB)`, `mlib_ImageDiv1_Fp_Inp(3MLIB)`, attributes(5)
The `mlib_ImageDivAlpha()` function divides color channels by the alpha channel on a pixel-by-pixel basis.

For the `MLIB_BYTE` image, it uses the following equation:
\[
dst[x][y][c] = \frac{src[x][y][c]}{\text{src}[x][y][a] \times 2^{(-8)}}
\]

For the `MLIB_SHORT` image, it uses the following equation:
\[
dst[x][y][c] = \frac{src[x][y][c]}{\text{src}[x][y][a] \times 2^{(-15)}}
\]

For the `MLIB_USHORT` image, it uses the following equation:
\[
dst[x][y][c] = \frac{src[x][y][c]}{\text{src}[x][y][a] \times 2^{(-16)}}
\]

For the `MLIB_INT` image, it uses the following equation:
\[
dst[x][y][c] = \frac{src[x][y][c]}{\text{src}[x][y][a] \times 2^{(-31)}}
\]

where \(c\) and \(a\) are the indices for the color channels and the alpha channel, respectively, so \(c \neq a\).

In the case of \(\text{src}[x][y][a] = 0\),
\[
\begin{align*}
dst[x][y][c] &= 0 & \text{if } & \text{src}[x][y][c] = 0 \\
dst[x][y][c] &= \text{DATA_TYPE_MAX} & \text{if } & \text{src}[x][y][c] > 0 \\
dst[x][y][c] &= \text{DATA_TYPE_MIN} & \text{if } & \text{src}[x][y][c] < 0
\end{align*}
\]

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

**Parameters**

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `cmask` Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

```c
#include <mlib.h>

mlib_status mlib_ImageDivAlpha(mlib_image *dst, const mlib_image *src, mlib_s32 cmask);
```
Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageDivAlpha_Inp(3MLIB), mlib_ImageDivAlpha_Fp(3MLIB), mlib_ImageDivAlpha_Fp_Inp(3MLIB), attributes(5)
The `mlib_ImageDivAlpha_Fp()` function divides floating-point color channels by the alpha channel on a pixel-by-pixel basis.

It uses the following equation:

\[
dst[x][y][c] = \frac{src[x][y][c]}{src[x][y][a]}
\]

where \(c\) and \(a\) are the indices for the color channels and the alpha channel, respectively, so \(c \neq a\).

The operation follows the IEEE-754 standard.

The function takes the following arguments:

- `dst`: Pointer to destination image.
- `src`: Pointer to source image.
- `cmask`: Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageDivAlpha(3MLIB), mlib_ImageDivAlpha_Fp_Inp(3MLIB), mlib_ImageDivAlpha_Inp(3MLIB), attributes(5)`
The `mlib_ImageDivAlpha_Fp_Inp()` function divides floating-point color channels by the alpha channel on a pixel-by-pixel basis, in place.

It uses the following equation:

\[ \text{srcdst}[x][y][c] = \frac{\text{srcdst}[x][y][c]}{\text{srcdst}[x][y][a]} \]

where \( c \) and \( a \) are the indices for the color channels and the alpha channel, respectively, so \( c \neq a \).

The operation follows the IEEE-754 standard.

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `cmask` Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageDivAlpha(3MLIB), mlib_ImageDivAlpha_Fp(3MLIB), mlib_ImageDivAlpha_Inp(3MLIB), attributes(5)`
### Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageDivAlpha_Inp(mlib_image *srcdst, mlib_s32 cmask);
```

### Description

The `mlib_ImageDivAlpha_Inp()` function divides color channels by the alpha channel on a pixel-by-pixel basis, in place.

For the `MLIB_BYTE` image, it uses the following equation:

\[
srcdst[x][y][c] = srcdst[x][y][c] / (srcdst[x][y][a] \times 2^{(-8)})
\]

For the `MLIB_SHORT` image, it uses the following equation:

\[
srcdst[x][y][c] = srcdst[x][y][c] / (srcdst[x][y][a] \times 2^{(-15)})
\]

For the `MLIB_USHORT` image, it uses the following equation:

\[
srcdst[x][y][c] = srcdst[x][y][c] / (srcdst[x][y][a] \times 2^{(-16)})
\]

For the `MLIB_INT` image, it uses the following equation:

\[
srcdst[x][y][c] = srcdst[x][y][c] / (srcdst[x][y][a] \times 2^{(-31)})
\]

where \(c\) and \(a\) are the indices for the color channels and the alpha channel, respectively, so \(c \neq a\).

In the case of \(srcdst[x][y][a] = 0\),

\[
\begin{align*}
srcdst[x][y][c] &= 0 & \text{if } srcdst[x][y][c] = 0 \\
srcdst[x][y][c] &= \text{DATA_TYPE_MAX} & \text{if } srcdst[x][y][c] > 0 \\
srcdst[x][y][c] &= \text{DATA_TYPE_MIN} & \text{if } srcdst[x][y][c] < 0
\end{align*}
\]

where \text{DATA_TYPE} is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

### Parameters

- `srcdst` Pointer to source and destination image.
- `cmask` Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:
### mlib_ImageDivAlpha_Inp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: `mlib_ImageDivAlpha(3MLIB), mlib_ImageDivAlpha_Fp(3MLIB), mlib_ImageDivAlpha_Fp_Inp(3MLIB), attributes(5)`
mlib_ImageDivConstShift – division by a constant, with shifting

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageDivConstShift(mlib_image *dst, const mlib_image *src, const mlib_s32 *c, mlib_s32 shift);

Description

The `mlib_ImageDivConstShift()` function divides each pixel in an image by a constant value on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

dst[x][y][i] = src[x][y][i] / c[i] * 2**shift

In the case of c[i] = 0,

dst[x][y][i] = 0 if src[x][y][i] = 0
dst[x][y][i] = DATA_TYPE_MAX if src[x][y][i] > 0
dst[x][y][i] = DATA_TYPE_MIN if src[x][y][i] < 0

where `DATA_TYPE` is `MLIB_U8`, `MLIB_S16`, `MLIB_U16`, or `MLIB_S32` for an image of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`, respectively.

Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `c` Constant by which each pixel is divided. c[i] contains the constant for channel i.
- `shift` Left shifting factor. 0 ≤ shift ≤ 31.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_ImageDivConstShift_Inp(3MLIB)`, attributes(5)
The `mlib_ImageDivConstShift_Inp` function divides each pixel in an image by a constant value on a pixel-by-pixel basis, in-place. It scales the result by a left shift and writes the result to the image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \frac{\text{srcdst}[x][y][i]}{c[i]} \times 2^{\text{shift}}
\]

In the case of \( c[i] = 0 \),

\[
\begin{align*}
\text{srcdst}[x][y][i] &= 0 & \text{if} & \quad \text{srcdst}[x][y][i] = 0 \\
\text{srcdst}[x][y][i] &= \text{DATA_TYPE_MAX} & \text{if} & \quad \text{srcdst}[x][y][i] > 0 \\
\text{srcdst}[x][y][i] &= \text{DATA_TYPE_MIN} & \text{if} & \quad \text{srcdst}[x][y][i] < 0
\end{align*}
\]

where \text{DATA_TYPE} is \text{MLIB_U8}, \text{MLIB_S16}, \text{MLIB_U16}, or \text{MLIB_S32} for an image of type \text{MLIB_BYTE}, \text{MLIB_SHORT}, \text{MLIB_USHORT}, or \text{MLIB_INT}, respectively.

**Parameters**  
The function takes the following arguments:

- `srcdst`  
  Pointer to source and destination image.

- `c`  
  Constant by which each pixel is divided. \( c[i] \) contains the constant for channel \( i \).

- `shift`  
  Left shifting factor. \( 0 \leq \text{shift} \leq 31 \).

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_ImageDivConstShift(3MLIB), attributes(5)`
The `mlib_ImageDiv_Fp()` function divides the second floating-point source image into the first floating-point source image on a pixel-by-pixel basis.

It uses the following equation:

\[ dst[x][y][i] = src1[x][y][i] / src2[x][y][i] \]

where the operation follows the IEEE-754 standard.

**Parameters**

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageDiv1_Fp_Inp(3MLIB), mlib_ImageDiv2_Fp_Inp(3MLIB), attributes(5)`
Name  mlib_ImageDivShift1_Inp – division with shifting, in place

Synopsis  
cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlibStatus mlib_ImageDivShift1_Inp(mlib_image *src1dst,
    const mlib_image *src2, mlib_s32 shift);

Description  The mlib_ImageDivShift1_Inp() function divides the second source image into the first source image on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[ src1dst[x][y][i] = src1dst[x][y][i] / src2[x][y][i] \times 2^{\text{shift}} \]

In the case of \( src2[x][y][i] = 0 \),

\[
\begin{align*}
src1dst[x][y][i] &= 0 & \text{if} \ src1dst[x][y][i] &= 0 \\
src1dst[x][y][i] &= \text{DATA\_TYPE\_MAX} & \text{if} \ src1dst[x][y][i] &= 0 \\
src1dst[x][y][i] &= \text{DATA\_TYPE\_MIN} & \text{if} \ src1dst[x][y][i] &= 0 \\
\end{align*}
\]

where \text{DATA\_TYPE} is mlib\_U8, mlib\_S16, mlib\_U16, or mlib\_S32 for an image of type mlib\_BYTE, mlib\_SHORT, mlib\_USHORT, or mlib\_INT, respectively.

Parameters  The function takes the following arguments:

- \text{src1dst}  Pointer to first source and destination image.
- \text{src2}  Pointer to second source image.
- \text{shift}  Left shifting factor. \( 0 \leq \text{shift} \leq 31 \).

Return Values  The function returns mlib\_SUCCESS if successful. Otherwise it returns mlib\_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib\_ImageDivShift(3MLIB), mlib\_ImageDivShift2\_Inp(3MLIB), attributes(5)
REFERENCE

Multimedia Library Functions - Part 3
Name  
mlib_ImageDivShift2_Inp – division with shifting, in place

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageDivShift2_Inp(mlib_image *src2dst,
        const mlib_image *src1, mlib_s32 shift);

Description  
The `mlib_ImageDivShift2_Inp()` function divides the second source image into the first source image on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[
sr2dst[x][y][i] = \frac{src1[x][y][i]}{sr2dst[x][y][i]} \times 2^{\text{shift}}
\]

In the case of \(sr2dst[x][y][i] = 0\),

\[
sr2dst[x][y][i] = 0 \quad \text{if} \quad sr1[x][y][i] = 0
\]

\[
sr2dst[x][y][i] = \text{DATA\_TYPE\_MAX} \quad \text{if} \quad sr1[x][y][i] > 0
\]

\[
sr2dst[x][y][i] = \text{DATA\_TYPE\_MIN} \quad \text{if} \quad sr1[x][y][i] < 0
\]

where DATA\_TYPE is MLIB\_U8, MLIB\_S16, MLIB\_U16, or MLIB\_S32 for an image of type MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT, or MLIB\_INT, respectively.

Parameters  
The function takes the following arguments:

- \(sr2dst\)  
  Pointer to second source and destination image.

- \(src1\)  
  Pointer to first source image.

- \(shift\)  
  Left shifting factor. \(0 \leq shift \leq 31\).

Return Values  
The function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib\_ImageDivShift(3MLIB), mlib\_ImageDivShift1\_Inp(3MLIB), attributes(5)
The `mlib_ImageDivShift()` function divides the second source image into the first source image on a pixel-by-pixel basis. It scales the result by a left shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[ dst[x][y][i] = \frac{src1[x][y][i]}{src2[x][y][i]} \times 2^{shift} \]

In the case of \( src2[x][y][i] = 0 \),

\[ dst[x][y][i] = \begin{cases} 0 & \text{if } src1[x][y][i] = 0 \\ DATA\_TYPE\_\text{MAX} & \text{if } src1[x][y][i] > 0 \\ DATA\_TYPE\_\text{MIN} & \text{if } src1[x][y][i] < 0 \end{cases} \]

where \( DATA\_TYPE \) is \( MLIB\_U8, MLIB\_S16, MLIB\_U16, \) or \( MLIB\_S32 \) for an image of type \( MLIB\_BYTE, MLIB\_SHORT, MLIB\_USHORT, \) or \( MLIB\_INT \), respectively.

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src1**: Pointer to first source image.
- **src2**: Pointer to second source image.
- **shift**: Left shifting factor. \( 0 \leq shift \leq 31 \).

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageDivShift1_Inp(3MLIB), mlib_ImageDivShift2_Inp(3MLIB), attributes(5)`
# mlib_ImageErode4

The `mlib_ImageErode4` function performs an erode operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 4-neighbor binary images, it uses the following equation:

\[
\text{dst}[x][y][0] = \text{AND}\{ \\text{src}[x][y][0], \\
\text{src}[x-1][y][0], \text{src}[x+1][y][0], \\
\text{src}[x][y-1][0], \text{src}[x][y+1][0] \}
\]

For 4-neighbor grayscale images, it uses the following equation:

\[
\text{dst}[x][y][0] = \text{MIN}\{ \\text{src}[x][y][0], \\
\text{src}[x-1][y][0], \text{src}[x+1][y][0], \\
\text{src}[x][y-1][0], \text{src}[x][y+1][0] \}
\]

where \(x = 1, \ldots, w-2\); \(y = 1, \ldots, h-2\).

## Parameters
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageErode4_Fp(3MLIB)`
- `mlib_ImageErode8(3MLIB)`
- `mlib_ImageErode8_Fp(3MLIB)`, attributes(5)
mlib_ImageErode4_Fp

Name
mlib_ImageErode4_Fp – four neighbor erode

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageErode4_Fp(mlib_image *dst, const mlib_image *src);

Description
The mlib_ImageErode4_Fp() function performs an erode operation on an image by using each pixel's four orthogonal neighbors. The source and destination images must be single-channel images. The data type of the images can be MLIB_FLOAT or MLIB_DOUBLE.

For 4-neighbor grayscale images, it uses the following equation:

dst[x][y][0] = MIN{ src[x][y][0],
src[x-1][y][0], src[x+1][y][0],
src[x][y-1][0], src[x][y+1][0] }  

where x = 1, ..., w-2; y = 1, ..., h-2.

Parameters
The function takes the following arguments:

dst  Pointer to destination image.
src  Pointer to source image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageErode4(3MLIB), mlib_ImageErode8(3MLIB), mlib_ImageErode8_Fp(3MLIB), attributes(5)
The `mlib_ImageErode8()` function performs an erode operation on an image by using all eight of each pixel's neighbors. The source and destination images must be single-channel images. The data type can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

For 8-neighbor binary images, it uses the following equation:

\[
\text{dst}[x][y][0] = \text{AND}\{ \text{src}[p][q][0], \quad \begin{align*}
x-1 & \leq p \leq x+1; \\
y-1 & \leq q \leq y+1
\end{align*}\}
\]

For 8-neighbor grayscale images, it uses the following equation:

\[
\text{dst}[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \quad \begin{align*}
x-1 & \leq p \leq x+1; \\
y-1 & \leq q \leq y+1
\end{align*}\}
\]

where \(x = 1, \ldots, w-2; \ y = 1, \ldots, h-2\).

### Parameters
The function takes the following arguments:
- \(\text{dst}\) Pointer to destination image.
- \(\text{src}\) Pointer to source image.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageErode4(3MLIB)`, `mlib_ImageErode4_Fp(3MLIB)`
- `mlib_ImageErode8_Fp(3MLIB)`, attributes(5)
```c
#include <mlib.h>

mlib_status mlib_ImageErode8_Fp(mlib_image *dst, const mlib_image *src);
```

The `mlib_ImageErode8_Fp()` function performs an erode operation on an image by using all eight of each pixel’s neighbors. The source and destination images must be single-channel images. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

For 8-neighbor grayscale images, it uses the following equation:

```
dst[x][y][0] = MIN{ src[p][q][0],
    x-1 ≤ p ≤ x+1; y-1 ≤ q ≤ y+1 }
```

where \(x = 1, \ldots, w-2; y = 1, \ldots, h-2\).

**Parameters**

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageErode4(3MLIB)`, `mlib_ImageErode4_Fp(3MLIB)`, `mlib_ImageErode8(3MLIB)`
- `attributes(5)`
The `mlib_ImageExp()` function computes the exponent of the image pixels. It uses the following equation:

\[ \text{dst}[x][y][i] = e^{\text{src}[x][y][i]} \]

### Parameters
- `dst` Pointer to destination image.
- `src` Pointer to source image.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageExp_Fp(3MLIB)`, `mlib_ImageExp_Fp_Inp(3MLIB)`, `mlib_ImageExp_Inp(3MLIB)`, `attributes(5)`
The `mlib_ImageExp_Fp` function computes the exponent of the floating-point image pixels.

It uses the following equation:

\[ dst[x][y][i] = e^{**src[x][y][i]} \]

### Parameters

- **dst** Pointer to destination image.
- **src** Pointer to source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageExp(3MLIB)`, `mlib_ImageExp_Fp_Inp(3MLIB)`, `mlib_ImageExp_Inp(3MLIB)`, attributes(5)
The `mlib_ImageExp_Fp_Inp()` function computes the exponent of the floating-point image pixels. It uses the following equation:

\[ \text{srcdst}[x][y][i] = e^{*\text{srcdst}[x][y][i]} \]

**Parameters**
The function takes the following arguments:

- `srcdst` Pointer to source and destination image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageExp(3MLIB), mlib_ImageExp_Fp(3MLIB), mlib_ImageExp_Inp(3MLIB), attributes(5)`
mlib_ImageExp_Inp – computes the exponent of the image pixels

Synopsis

cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageExp_Inp(mlib_image *srcdst);

Description

The mlib_ImageExp_Inp() function computes the exponent of the image pixels, in place.

It uses the following equation:

\[ \text{srcdst}[x][y][i] = e^{\text{srcdst}[x][y][i]} \]

Parameters

srcdst Pointer to source and destination image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageExp(3MLIB), mlib_ImageExp_Fp(3MLIB), mlib_ImageExp_Fp_Inp(3MLIB), attributes(5)
Name mlib_ImageExtrema2, mlib_ImageExtrema2_Fp – image extrema

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageExtrema2(mlib_s32 *min, mlib_s32 *max,
    const mlib_image *img, mlib_s32 xStart, mlib_s32 yStart,
    mlib_s32 xPeriod, mlib_s32 yPeriod);

mlib_status mlib_ImageExtrema2_Fp(mlib_d64 *min, mlib_d64 *max,
    const mlib_image *img, mlib_s32 xStart, mlib_s32 yStart,
    mlib_s32 xPeriod, mlib_s32 yPeriod);

Description Each of the functions determines the extrema values for each channel in an image, possibly with subsampling.

It uses the following equation:

min[i] = MIN{ img[x][y][i] }
max[i] = MAX{ img[x][y][i] }

where

x = xStart + p*xPeriod; 0 \leq p < (w - xStart)/xPeriod
y = yStart + q*yPeriod; 0 \leq q < (h - yStart)/yPeriod

Parameters Each of the functions takes the following arguments:

- **min** Pointer to minimum vector, where length is the number of channels in the image. min[i] contains the minimum of channel i.
- **max** Pointer to maximum vector, where length is the number of channels in the image. max[i] contains the maximum of channel i.
- **img** Pointer to a source image.
- **xStart** Initial X sample coordinate.
- **yStart** Initial Y sample coordinate.
- **xPeriod** X sample rate. xPeriod \geq 1.
- **yPeriod** Y sample rate. yPeriod \geq 1.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
**mlib_ImageExtrema2(3MLIB)**

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageExtremaLocations(3MLIB), mlib_ImageMaximum(3MLIB),  
mlib_ImageMaximum_Fp(3MLIB), mlib_ImageMinimum(3MLIB),  
mlib_ImageMinimum_Fp(3MLIB), attributes(5)
mlib_ImageExtremaLocations(3MLIB)

Name
mlib_ImageExtremaLocations, mlib_ImageExtremaLocations_Fp – image extrema and their locations

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageExtremaLocations( mlib_s32 *min, mlib_s32 *max,
const mlib_image *img, mlib_s32 xStart, mlib_s32 yStart,
 mlib_s32 xPeriod, mlib_s32 yPeriod, mlib_s32 saveLocations,
 mlib_s32 maxRuns, mlib_s32 *minCounts, mlib_s32 *maxCounts,
 mlib_s32 **minLocations, mlib_s32 **maxLocations, mlib_s32 len);

mlib_status mlib_ImageExtremaLocations_Fp( mlib_d64 *min, mlib_d64 *max,
const mlib_image *img, mlib_s32 xStart, mlib_s32 yStart,
 mlib_s32 xPeriod, mlib_s32 yPeriod, mlib_s32 saveLocations,
 mlib_s32 maxRuns, mlib_s32 *minCounts, mlib_s32 *maxCounts,
 mlib_s32 **minLocations, mlib_s32 **maxLocations, mlib_s32 len);

Description
Each of the functions finds the image-wise minimum and maximum pixel values for each channel, and optionally, their locations.

Each of the functions scans an image, finds the minimum and maximum pixel values for each channel, and finds the locations of those pixels with the minimum or maximum values.

The user provides initial minimum/maximum values through the arguments min and max. This function will update them based on findings.

The set of pixels scanned may furthermore be reduced by specifying xPeriod and yPeriod parameters that specify the sampling rate along each axis.

The set of pixels to be scanned may be obtained from the following equation:

\[
x = x_{Start} + p \times x_{Period}; \quad 0 \leq p < (w - x_{Start})/x_{Period}
\]
\[
y = y_{Start} + q \times y_{Period}; \quad 0 \leq q < (h - y_{Start})/y_{Period}
\]

The locations of the minimum/maximum, if asked, are recorded in a format of run-length coding. Each run-length code, or simply called a run, has a format of \((x_{Start}, y_{Start}, length)\). Here length is defined on the low-resolution image (with downsampling factors of \(1/x_{Period}, 1/y_{Period}\)) and does not cross rows. So the run-length code \((x_{Start}, y_{Start}, length)\) means that the pixels at \((x_{Start}, y_{Start}), (x_{Start} + x_{Period}, y_{Start}), \ldots, (x_{Start} + (length - 1)\times x_{Period}, y_{Start})\) of the original image have a value of the minimum/maximum.

The buffers for minLocations and maxLocations are organized in the following format for each channel i:

\[
\text{minLocations}[i][0] = x_{Start0}; \quad \text{the 1st run}
\]
\[
\text{minLocations}[i][1] = y_{Start0};
\]
\[
\text{minLocations}[i][2] = length0;
\]
\[
\text{minLocations}[i][3] = x_{Start1}; \quad \text{the 2nd run}
\]
It is the user’s responsibility to allocate enough memory for the buffers for `minLocations` and `maxLocations`. This function may return `MLIB_OUTOFRANGE`, if any of the buffers is not big enough.

**Parameters**  
The function takes the following arguments:

- `min`: Pointer to the minimum values.
- `max`: Pointer to the maximum values.
- `img`: Pointer to the input image.
- `xStart`: Initial X sample coordinate.
- `yStart`: Initial Y sample coordinate.
- `xPeriod`: X sampling rate. `xPeriod ≥ 1`.
- `yPeriod`: Y sampling rate. `yPeriod ≥ 1`.
- `saveLocations`: If true (i.e., `saveLocations != 0`), find the extrema locations; otherwise only find the extrema.
- `maxRuns`: Number of runs of the minimum/maximum the caller expects for each channel. `maxRuns ≥ 1`. If it is `MLIB_S32_MAX`, all the minimum/maximum locations should be recorded.
- `minCounts`: Pointer to the numbers of runs of the minimum recorded in `minLocations`.
- `maxCounts`: Pointer to the numbers of runs of the maximum recorded in `maxLocations`.
- `minLocations`: Pointer to the minimum locations in a format of run-length coding.
- `maxLocations`: Pointer to the maximum locations in a format of run-length coding.
- `len`: Length of the buffers for the minimum/maximum locations in each channel.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

- mlib_ImageExtrema2(3MLIB)
- mlib_ImageMaximum(3MLIB)
- mlib_ImageMaximum_Fp(3MLIB)
- mlib_ImageMinimum(3MLIB)
- mlib_ImageMinimum_Fp(3MLIB)
- attributes(5)
mlib_ImageFilteredSubsample(3MLIB)

Name mlib_ImageFilteredSubsample, mlib_ImageFilteredSubsample_Fp – antialias filters and subsamples an image

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageFilteredSubsample(mlib_image *dst,
    const mlib_image *src, mlib_s32 scaleX, mlib_s32 scaleY,
    mlib_s32 transX, mlib_s32 transY,
    const mlib_d64 *hKernel, const mlib_d64 *vKernel, mlib_s32 hSize,
    mlib_s32 vSize, mlib_s32 hParity, mlib_s32 vParity, mlib_edge edge);

mlib_status mlib_ImageFilteredSubsample_Fp(mlib_image *dst,
    const mlib_image *src, mlib_s32 scaleX, mlib_s32 scaleY,
    mlib_s32 transX, mlib_s32 transY,
    const mlib_d64 *hKernel, const mlib_d64 *vKernel, mlib_s32 hSize,
    mlib_s32 vSize, mlib_s32 hParity, mlib_s32 vParity, mlib_edge edge);

Description Each of the functions antialias filters and subsamples an image.

The effect of one of the functions on an image is equivalent to performing convolution (filter) followed by subsampling (zoom out).

The functions are similar to the mlib_ImageZoomTranslate() and mlib_ImageZoomTranslate_Fp() functions. But they have different definitions on scale factors and translations, hence use different coordinate mapping equations. The scaleX and scaleY used by mlib_ImageFilteredSubsample() and mlib_ImageFilteredSubsample_Fp() are the reciprocals of the zoomx and zoomy, respectively, used by mlib_ImageZoomTranslate() and mlib_ImageZoomTranslate_Fp().

The functions use the following equations for coordinate mapping:

\[
x_S = x_D * \text{scaleX} + \text{transX}
\]
\[
y_S = y_D * \text{scaleY} + \text{transY}
\]

where, a point \((x_D, y_D)\) in the destination image is backward mapped to a point \((x_S, y_S)\) in the source image. The arguments transX and transY are provided to support tiling.

The subsample terms, i.e., the scale factors scaleX and scaleY, are restricted to positive integral values. Geometrically, one destination pixel maps to scaleX by scaleY source pixels. With odd scale factors, destination pixel centers map directly onto source pixel centers. With even scale factors, destination pixel centers map squarely between source pixel centers. Below are examples of even, odd, and combination cases.

```
  s s s s s s  s s s s s s
  d d       d d
  s s s s s s  s d s s d s
  s s s s s s  s d s s d s
  d d       d d
```
The applied filter is quadrant symmetric (typically antialias + resample). The filter is product-separable, quadrant symmetric, and is defined by half of its span. Parity is used to signify whether the symmetric kernel has a double center (even parity) or a single center value (odd parity). For example, if \( h_{\text{Parity}} = 0 \) (even), the horizontal kernel is defined as:

\[
\text{hKernel}[\text{hSize}-1], \ldots, \text{hKernel}[0], \text{hKernel}[0], \ldots, \text{hKernel}[\text{hSize}-1]
\]

Otherwise, if \( h_{\text{Parity}} = 1 \) (odd), the horizontal kernel is defined as:

\[
\text{hKernel}[\text{hSize}-1], \ldots, \text{hKernel}[0], \ldots, \text{hKernel}[\text{hSize}-1]
\]

Horizontal and vertical kernels representing convolved resample (i.e., the combined separable kernels) can be computed from a convolution filter (with odd parity), a resample filter, and because the subsample factors affect resample weights, the subsample scale factors. It is the user's responsibility to provide meaningful combined kernels.

To compute the value of a pixel centered at point \((x_D, y_D)\) in the destination image, apply the combined kernel to the source image by aligning the kernel's geometric center to the backward mapped point \((x_S, y_S)\) in the source image. In the cases that it can not be exactly
on top of point \((x_S, y_S)\), the kernel’s center should be half-pixel right and/or below that point. When this is done in a separable manner, the centers of horizontal and vertical kernels should align with \(x_S\) and \(y_S\), respectively.

The combination of subsampling and filtering has performance benefits over sequential function usage in part due to the symmetry constraints imposed by only allowing integer parameters for scaling and only allowing separable symmetric filters.

**Parameters**
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **scaleX** The x scale factor of subsampling.
- **scaleY** The y scale factor of subsampling.
- **transX** The x translation.
- **transY** The y translation.
- **hKernel** Pointer to the compact form of horizontal kernel.
- **vKernel** Pointer to the compact form of vertical kernel.
- **hSize** Size of array **hKernel**.
- **vSize** Size of array **vKernel**.
- **hParity** Parity of horizontal kernel (0: even, 1: odd).
- **vParity** Parity of vertical kernel (0: even, 1: odd).
- **edge** Type of edge condition. It can be one of the following:
  - **MLIB_EDGE_DST_NO_WRITE**

**Return Values**
The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**
See **attributes(5)** for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- **mlib_ImageFilteredSubsample(3MLIB)**
- **mlib_ImageZoomTranslate(3MLIB)**
- **mlib_ImageZoomTranslate_Fp(3MLIB)**
- **attributes(5)**
# mlib_ImageFlipAntiDiag

## Name
mlib_ImageFlipAntiDiag – anti-diagonal flip

## Synopsis
```c
#include <mlib.h>

mlib_status mlib_ImageFlipAntiDiag(mlib_image *dst, const mlib_image *src);
```

## Description
The `mlib_ImageFlipAntiDiag()` function flips an image on the anti-diagonal axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

## Parameters
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageFlipAntiDiag_Fp(3MLIB)`
- `mlib_ImageFlipMainDiag(3MLIB)`
- `mlib_ImageFlipMainDiag_Fp(3MLIB)`
- `mlib_ImageFlipX(3MLIB)`
- `mlib_ImageFlipX_Fp(3MLIB)`
- `mlib_ImageFlipY(3MLIB)`
- `mlib_ImageFlipY_Fp(3MLIB)`
- `mlib_ImageRotate90(3MLIB)`
- `mlib_ImageRotate90_Fp(3MLIB)`
- `mlib_ImageRotate180(3MLIB)`
- `mlib_ImageRotate180_Fp(3MLIB)`
- `mlib_ImageRotate270(3MLIB)`
- `mlib_ImageRotate270_Fp(3MLIB)`
- `attributes(5)`
The `mlib_ImageFlipAntiDiag_Fp()` function flips a floating-point image on the anti-diagonal axis. The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB), mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB), mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB), mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB), mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)`
mlib_ImageFlipMainDiag(3MLIB)

Name  mlib_ImageFlipMainDiag – main diagonal flip

Synopsis  cc [ flag ... ] file... -tlib [ library ... ]
          #include <mlib.h>

          mlib_status mlib_ImageFlipMainDiag(mlib_image *dst, const mlib_image *src);

Description  The mlib_ImageFlipMainDiag() function flips an image on the main diagonal.

The width and height of the destination image can be different from the width and height of
the source image. The center of the source image is mapped to the center of the destination
image.

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or
MLIB_INT.

Parameters  The function takes the following arguments:

          dst  Pointer to destination image.

          src  Pointer to source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag_Fp(3MLIB), mlib_ImageFlipX(3MLIB),
mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB), mlib_ImageFlipY_Fp(3MLIB),
mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
The `mlib_ImageFlipMainDiag_Fp()` function flips a floating-point image on the main diagonal.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

### Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB), mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB), mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB), mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB), mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB)`
mlib_ImageFlipX–X-axis flip

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageFlipX(mlib_image *dst, const mlib_image *src);

Description
The mlib_ImageFlipX() function flips an image on its X axis.

The width and height of the destination image can be different from the width and height of
the source image. The center of the source image is mapped to the center of the destination
image.

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or
MLIB_INT.

Parameters
The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB), mlib_ImageFlipY_Fp(3MLIB),
mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
# mlib_ImageFlipX_Fp

**Name**  
mlib_ImageFlipX_Fp – X-axis flip

**Synopsis**  
```c
#include <mlib.h>

mlib_status mlib_ImageFlipX_Fp(mlib_image *dst, const mlib_image *src);
```

**Description**  
The `mlib_ImageFlipX_Fp()` function flips a floating-point image on its X axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

**Parameters**  
The function takes the following arguments:
- `dst`  
  Pointer to destination image.
- `src`  
  Pointer to source image.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipY(3MLIB), mlib_ImageFlipY_Fp(3MLIB),
mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)`
mlib_ImageFlipY – Y-axis flip

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageFlipY(mlib_image *dst, const mlib_image *src);

Description

The mlib_ImageFlipY() function flips an image on its Y axis.

The width and height of the destination image can be different from the width and height of
the source image. The center of the source image is mapped to the center of the destination
image.

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or
MLIB_INT.

Parameters

The function takes the following arguments:

- **dst**  Pointer to destination image.
- **src**  Pointer to source image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY_Fp(3MLIB),
mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
**mlib_ImageFlipY_Fp**

**Name**  
mlib_ImageFlipY_Fp – Y-axis flip

**Synopsis**  
`cc [ flag... ] file... -lmlib [ library... ]`  
#include `<mlib.h>`

```c
mlib_status mlib_ImageFlipY_Fp(mlib_image *dst, const mlib_image *src);
```

**Description**  
The `mlib_ImageFlipY_Fp()` function flips a floating-point image on its Y axis.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

**Parameters**  
The function takes the following arguments:

- `dst`  
  Pointer to destination image.

- `src`  
  Pointer to source image.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB),
mlib_ImageRotate90(3MLIB), mlib_ImageRotate90_Fp(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
The `mlib_ImageFourierTransform()` function performs a two-dimensional Fourier transformation. The source and destination images must be the same type and the same size. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`. The height and width of the images must be some positive power of 2 (but they do not have to be equal).

They can have 1 or 2 channels. If the source image has just one channel the imaginary parts are assumed to be zero. If the destination image has just one channel, then it is assumed that the imaginary parts of the output can be discarded. But in case both source and destination images are one-channel images, then `MLIB_FAILURE` is returned.

The predefined modes used in the image Fourier transform function are as follows:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_DFT_SCALE_NONE</td>
<td>Forward DFT without scaling</td>
</tr>
<tr>
<td>MLIB_DFT_SCALE_MXN</td>
<td>Forward DFT with scaling of (1/(M\times N))</td>
</tr>
<tr>
<td>MLIB_DFT_SCALE_SQRT</td>
<td>Forward DFT with scaling of (1/\sqrt{M\times N})</td>
</tr>
<tr>
<td>MLIB_IDFT_SCALE_NONE</td>
<td>Inverse DFT without scaling</td>
</tr>
<tr>
<td>MLIB_IDFT_SCALE_MXN</td>
<td>Inverse DFT with scaling of (1/(M\times N))</td>
</tr>
<tr>
<td>MLIB_IDFT_SCALE_SQRT</td>
<td>Inverse DFT with scaling of (1/\sqrt{M\times N})</td>
</tr>
</tbody>
</table>

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `mode` Mode of the transform.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
### ATTRIBUTE TYPE
### ATTRIBUTE VALUE

| MT-Level | MT-Safe |

**See Also** [attributes(5)]
Name  mlib_ImageGetBitOffset – get bitoffset

Synopsis  cc [ flag...] file... -lmlib [ library... ]
#include <mlib.h>

mlib_s32 mlib_ImageGetBitOffset(const mlib_image *img);

Description  A query function that returns the bitoffset public field of a mediaLib image structure. The data
type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT,
MLIB_FLOAT, or MLIB_DOUBLE.

Parameters  The function takes the following arguments:

  img  Pointer to a mediaLib image structure.

Return Values  The function returns the offset, in terms of bits, of an image from the beginning of the data
buffer to the first pixel.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageGetChannels(3MLIB), mlib_ImageGetData(3MLIB),
mlib_ImageGetFlags(3MLIB), mlib_ImageGetHeight(3MLIB),
mlib_ImageGetPaddings(3MLIB), mlib_ImageGetStride(3MLIB),
mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)
**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_s32 mlib_ImageGetChannels(const mlib_image *img);
```

**Description**

A query function that returns the channels public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

**Parameters**

The function takes the following arguments:

- `img` Pointer to a mediaLib image structure.

**Return Values**

The function returns the number of channels in an image.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetData(3MLIB), mlib_ImageGetFlags(3MLIB), mlib_ImageGetHeight(3MLIB), mlib_ImageGetPaddings(3MLIB), mlib_ImageGetStride(3MLIB), mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)
The `mlib_ImageGetData()` function returns the data public field of a mediaLib image structure. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

The function takes the following arguments:

- `img` -- Pointer to source image.

The function returns a pointer to the image data.

See also `mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetChannels(3MLIB), mlib_ImageGetFlags(3MLIB), mlib_ImageGetHeight(3MLIB), mlib_ImageGetPaddings(3MLIB), mlib_ImageGetStride(3MLIB), mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)`
### mlib_ImageGetFlags

**Synopsis**

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_s32 mlib_ImageGetFlags(const mlib_image *img);
```

**Description**
The `mlib_ImageGetFlags()` function returns the attribute flags of an image. The data type of the image can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

**Parameters**
The function takes the following arguments:
- `img` Pointer to source image.

**Return Values**
The function returns the value of the attribute flags.

**Attributes**
See [attributes](5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageGetBitOffset(3MLIB)`, `mlib_ImageGetChannels(3MLIB)`, `mlib_ImageGetData(3MLIB)`, `mlib_ImageGetHeight(3MLIB)`, `mlib_ImageGetPaddings(3MLIB)`, `mlib_ImageGetStride(3MLIB)`, `mlib_ImageGetType(3MLIB)`, `mlib_ImageGetWidth(3MLIB)`, [attributes](5)
Name
mlib_ImageGetFormat – get format

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_format mlib_ImageGetFormat(const mlib_image *img);

Description
A query function that returns the format public field of a mlib_image structure. The data type of
the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT,
or MLIB_DOUBLE.

Parameters
The function takes the following arguments:

img Pointer to a mediaLib image structure.

Return Values
The function returns the value of the format of an image.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageSetFormat(3MLIB), mlib_ImageGetBitOffset(3MLIB),
mlib_ImageGetChannels(3MLIB), mlib_ImageGetData(3MLIB),
mlib_ImageGetFlags(3MLIB), mlib_ImageGetHeight(3MLIB),
mlib_ImageGetPaddings(3MLIB), mlib_ImageGetStride(3MLIB),
mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)
**Name**  
mlib_ImageGetHeight – get height

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```
mlib_s32 mlib_ImageGetHeight(const mlib_image *img);
```

**Description**  
A query function that returns the height public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

**Parameters**  
The function takes the following arguments:

- `img`  
  Pointer to source image.

**Return Values**  
The function returns the value of the height (in pixels) of an image.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetChannels(3MLIB), mlib_ImageGetData(3MLIB), mlib_ImageGetFlags(3MLIB), mlib_ImageGetPaddings(3MLIB), mlib_ImageGetStride(3MLIB), mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)
mlib_ImageGetPaddings - get paddings

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_u8 *mlib_ImageGetPaddings(const mlib_image *img);

Description
A query function that returns the borders public field of a multimediaLib image structure. The data
type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT,
MLIB_FLOAT, or MLIB_DOUBLE.

Parameters
The function takes the following arguments:

img Pointer to a multimediaLib image structure.

Return Values
The function returns a pointer to the image paddings. paddings[0] holds leftPadding;
bottomPadding.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetChannels(3MLIB),
mlib_ImageGetData(3MLIB), mlib_ImageGetFlags(3MLIB),
mlib_ImageGetHeight(3MLIB), mlib_ImageGetStride(3MLIB),
mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB),
mlib_ImageSetPaddings(3MLIB), attributes(5)
**Name**  
mlib_ImageGetStride – get stride

**Synopsis**  
cc [ flag... ] file... -lmllib [ library... ]  
#include <mlib.h>

```c
mlib_s32 mlib_ImageGetStride(const mlib_image *img);
```

**Description**  
A query function that returns the stride public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

**Parameters**  
The function takes the following arguments:

- `img`  
  Pointer to source image.

**Return Values**  
The function returns the value of the stride (in bytes) of an image.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetChannels(3MLIB), mlib_ImageGetData(3MLIB), mlib_ImageGetFlags(3MLIB), mlib_ImageGetHeight(3MLIB), mlib_ImageGetPaddings(3MLIB), mlib_ImageGetType(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)
Name mlib_ImageGetType – get type

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_type mlib_ImageGetType(const mlib_image *img);

Description A query function that returns the type public field of a mediaLib image structure. The data
type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT,
MLIB_FLOAT, or MLIB_DOUBLE.

Parameters The function takes the following arguments:

   img Pointer to source image.

Return Values The function returns the value of the type of an image.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetChannels(3MLIB),
mlib_ImageGetData(3MLIB), mlib_ImageGetFlags(3MLIB),
mlib_ImageGetHeight(3MLIB), mlib_ImageGetPaddings(3MLIB),
mlib_ImageGetStride(3MLIB), mlib_ImageGetWidth(3MLIB), attributes(5)
### Name
mlib_ImageGetWidth – get width

### Synopsis
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_s32 mlib_ImageGetWidth(const mlib_image *img);
```

### Description
A query function that returns the width public field of a mediaLib image structure. The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

### Parameters
The function takes the following arguments:

- **img**: Pointer to source image.

### Return Values
The function returns the value of the width of an image.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
mlib_ImageGetBitOffset(3MLIB), mlib_ImageGetChannels(3MLIB),
mlib_ImageGetData(3MLIB), mlib_ImageGetFlags(3MLIB),
mlib_ImageGetHeight(3MLIB), mlib_ImageGetPaddings(3MLIB),
mlib_ImageGetStride(3MLIB), mlib_ImageGetType(3MLIB), attributes(5)
The `mlib_ImageGradient3x3` function performs edge detection by computing the magnitude of the image gradient vector in two orthogonal directions using 3x3 gradient filtering.

It uses the following equation:
\[
dst[x][y][i] = ( SH(x,y,i)**2 + SV(x,y,i)**2 )**0.5
\]

where `SH()` and `SV()` are the horizontal and vertical gradient images generated from the corresponding channel of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks.

### Parameters

The function takes the following arguments:

- **`dst`** Pointer to destination image.
- **`src`** Pointer to source image.
- **`hmask`** Pointer to horizontal mask in row-major order.
- **`vmask`** Pointer to vertical mask in row-major order.
- **`cmask`** Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
- **`edge`** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SR_EXTEND`

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_ImageGradient3x3_Fp(3MLIB), mlib_ImageGradientMxN(3MLIB),
mlib_ImageGradientMxN_Fp(3MLIB), attributes(5)
The `mlib_ImageGradient3x3_Fp()` function performs floating-point edge detection by computing the magnitude of the image gradient vector in two orthogonal directions using 3x3 gradient filtering.

It uses the following equation:

\[
\text{dst}[x][y][i] = ( \text{SH}(x,y,i)^2 + \text{SV}(x,y,i)^2 )^{0.5}
\]

where \(\text{SH}()\) and \(\text{SV}()\) are the horizontal and vertical gradient images generated from the corresponding channel of the source image by correlating it with the supplied orthogonal (horizontal and vertical) gradient masks.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `hmask` Pointer to horizontal mask in row-major order.
- `vmask` Pointer to vertical mask in row-major order.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DS_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SR_EXTEND`

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also: mlib_ImageGradient3x3(3MLIB), mlib_ImageGradientMxN(3MLIB), mlib_ImageGradientMxN_Fp(3MLIB), attributes(5)
Name  
mlib_ImageGradientMxN – MxN gradient filter

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageGradientMxN(mlib_image *dst, const mlib_image *src,
const mlib_d64 *hmask, const mlib_d64 *vmask, mlib_s32 m, mlib_s32 n,
mlib_s32 dm, mlib_s32 dn, mlib_s32 cmask, mlib_edge edge);

Description  
The mlib_ImageGradientMxN() function performs edge detection by computing the
magnitude of the image gradient vector in two orthogonal directions using MxN gradient
filtering.

It uses the following equation:

dst[x][y][i] = ( SH(x,y,i)**2 + SV(x,y,i)**2 )**0.5

where SH() and SV() are the horizontal and vertical gradient images generated from the
corresponding channel of the source image by correlating it with the supplied orthogonal
(horizontal and vertical) gradient masks.

Parameters  
The function takes the following arguments:

dst  Pointer to destination image.
src  Pointer to source image.
hmask  Pointer to horizontal mask in row-major order.
vmask  Pointer to vertical mask in row-major order.
m  Width of the convolution kernel. m > 1.
n  Height of the convolution kernel. n > 1.
dm  X coordinate of the key element in the convolution kernel. 0 ≤ dm < m.
dn  Y coordinate of the key element in the convolution kernel. 0 ≤ dn < n.
cmask  Channel mask to indicate the channels to be convolved, each bit of which
represents a channel in the image. The channels corresponding to 1 bits are those
to be processed. For a single channel image, the channel mask is ignored.

edge  Type of edge condition. It can be one of the following:

MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DS_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SR_EXTEND
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_ImageGradientMxN_Fp(3MLIB), mlib_ImageGradient3x3(3MLIB), mlib_ImageGradient3x3_Fp(3MLIB), attributes(5)`
mlib_ImageGradientMxN_Fp

Name
mlib_ImageGradientMxN_Fp – MxN gradient filter

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include "mlib.h"

mlib_status mlib_ImageGradientMxN_Fp(mlib_image *dst,
    const mlib_image *src, const mlib_d64 *hmask,
    const mlib_d64 *vmask, mlib_s32 m, mlib_s32 n, mlib_s32 dm,
    mlib_s32 dn, mlib_s32 cmask, mlib_edge edge);

Description
The mlib_ImageGradientMxN_Fp() function performs floating-point edge detection by
computing the magnitude of the image gradient vector in two orthogonal directions using
MxN gradient filtering.

It uses the following equation:

dst[x][y][i] = ( SH(x,y,i)**2 + SV(x,y,i)**2 )**0.5

where SH() and SV() are the horizontal and vertical gradient images generated from the
corresponding channel of the source image by correlating it with the supplied orthogonal
(horizontal and vertical) gradient masks.

Parameters
The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
hmask Pointer to horizontal mask in row-major order.
vmask Pointer to vertical mask in row-major order.
m Width of the convolution kernel. m > 1.
n Height of the convolution kernel. n > 1.
dm X coordinate of the key element in the convolution kernel. 0 ≤ dm < m.
dn Y coordinate of the key element in the convolution kernel. 0 ≤ dn < n.
cmask Channel mask to indicate the channels to be convolved, each bit of which
represents a channel in the image. The channels corresponding to 1 bits are those
to be processed. For a single channel image, the channel mask is ignored.
edge Type of edge condition. It can be one of the following:

MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DS_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SR_EXTEND
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageGradientMxN(3MLIB), mlib_ImageGradient3x3(3MLIB), mlib_ImageGradient3x3_Fp(3MLIB), attributes(5)
mlib_ImageGridWarp–grid-based imagewarp

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_ImageGridWarp(mlib_image *dst, const mlib_image *src,  
const mlib_f32 *xWarpPos, const mlib_f32 *yWarpPos,  
mlib_d64 postShiftX, mlib_d64 postShiftY,  
mlib_s32 xStart, mlib_s32 xStep, mlib_s32 xNumCells,  
mlib_s32 yStart, mlib_s32 yStep, mlib_s32 yNumCells,  
mlib_filter filter, mlib_edge edge);

Description  
The mlib_ImageGridWarp() function performs a regular grid-based image warp. The images
must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4
channels. The data type of the images can be MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or
MLIB_INT. The two images may have different sizes.

The image pixels are assumed to be centered at \( \frac{1}{2} \) coordinate points. For example, the
upper-left corner pixel of an image is located at \((0.5, 0.5)\).

For each pixel in the destination image, its center point \( D \) is, first, backward mapped to a point
\( S \) in the source image; then the source pixels with their centers surrounding point \( S \) are
selected to do one of the interpolations specified by the filter parameter to generate the pixel
value for point \( D \).

The mapping from destination pixels to source positions is described by bilinear interpolation
between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate \((x, y)\) that lies within a cell having corners at \((x_0, y_0)\),
\((x_1, y_0)\), \((x_0, y_1)\) and \((x_1, y_1)\), with source coordinates defined at each respective corner
equal to \((sx_0, sy_0)\), \((sx_1, sy_1)\), \((sx_2, sy_2)\) and \((sx_3, sy_3)\), the source position \((sx, sy)\)
that maps onto \((x, y)\) is given by the formulas:

\[
\frac{x}{x} = \frac{x - x_0}{x_1 - x_0} \quad \frac{y}{x} = \frac{y - y_0}{y_1 - y_0}
\]

\[
s = sx_0 + (sx_1 - sx_0)xfrac
\]

\[
t = sy_0 + (sy_1 - sy_0)xfrac
\]

\[
u = sx_2 + (sx_3 - sx_2)xfrac
\]

\[
v = sy_2 + (sy_3 - sy_2)xfrac
\]

\[
sx = s + (u - s)yfrac - postShiftX
\]

\[
sy = t + (v - t)yfrac - postShiftY
\]

In other words, the source \( x \) and \( y \) values are interpolated horizontally along the top and
bottom edges of the grid cell, and the results are interpolated vertically:
The results of above interpolation are shifted by \((-\text{postShiftX}, -\text{postShiftY})\) to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at \((x_{\text{start}}, y_{\text{start}})\). Each cell has width equal to \(x_{\text{step}}\) and height equal to \(y_{\text{step}}\), and there are \(x_{\text{numCells}}\) cells horizontally and \(y_{\text{numCells}}\) cells vertically.

The degree of warping within each cell is defined by the values in \(x_{\text{warpPos}}\) and \(y_{\text{warpPos}}\) parameters. Each of these parameters must contain \((x_{\text{numCells}} + 1) \times (y_{\text{numCells}} + 1)\) values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose \(x_{\text{numCells}}\) is equal to 2 and \(y_{\text{numCells}}\) is equal to 1. Then the order of the data in the \(x_{\text{warpPos}}\) would be:

\[
x_{00}, x_{10}, x_{20}, x_{01}, x_{11}, x_{21}
\]

and in the \(y_{\text{warpPos}}\):

\[
y_{00}, y_{10}, y_{20}, y_{01}, y_{11}, y_{21}
\]

for a total of \((2 + 1) \times (1 + 1) = 6\) elements in each table.

**Parameters**

The function takes the following arguments:

- \(dst\) Pointer to destination image.
- \(src\) Pointer to source image.
xWarpPos  A float array of length \((x\text{NumCells} + 1)*(y\text{NumCells} + 1)\) containing horizontal warp positions at the grid points, in row-major order.

yWarpPos  A float array of length \((x\text{NumCells} + 1)*(y\text{NumCells} + 1)\) containing vertical warp positions at the grid points, in row-major order.

postShiftX  The displacement to apply to source X positions.

postShiftY  The displacement to apply to source Y positions.

xStart  The minimum X coordinate of the grid.

xStep  The horizontal spacing between grid cells.

xNumCells  The number of grid cell columns.

yStart  The minimum Y coordinate of the grid.

yStep  The vertical spacing between grid cells.

yNumCells  The number of grid cell rows.

filter  Type of resampling filter. It can be one of the following:

MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2

edge  Type of edge condition. It can be one of the following:

MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_SRC_PADDED

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageGridWarp_Fp(3MLIB), mlib_ImageGridWarpTable(3MLIB), mlib_ImageGridWarpTable_Fp(3MLIB), attributes(5)
The `mlib_ImageGridWarp_Fp()` function performs a regular grid-based image warp on a floating-point image. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image, then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate \((x, y)\) that lies within a cell having corners at \((x0, y0)\), \((x1, y0)\), \((x0, y1)\) and \((x1, y1)\), with source coordinates defined at each respective corner equal to \((sx0, sy0)\), \((sx1, sy1)\), \((sx2, sy2)\) and \((sx3, sy3)\), the source position \((sx, sy)\) that maps onto \((x, y)\) is given by the formulas:

\[
\begin{align*}
  xfrac &= (x - x0)/(x1 - x0) \\
  yfrac &= (y - y0)/(y1 - y0) \\
  s &= sx0 + (sx1 - sx0)*xfrac \\
  t &= sy0 + (sy1 - sy0)*xfrac \\
  u &= sx2 + (sx3 - sx2)*xfrac \\
  v &= sy2 + (sy3 - sy2)*xfrac \\
  sx &= s + (u - s)*yfrac - postShiftX \\
  sy &= t + (v - t)*yfrac - postShiftY
\end{align*}
\]

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically.
The results of above interpolation are shifted by \((-\text{postShiftX}, -\text{postShiftY})\) to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at \((x\text{Start}, y\text{Start})\). Each cell has width equal to \(x\text{Step}\) and height equal to \(y\text{Step}\), and there are \(x\text{NumCells}\) cells horizontally and \(y\text{NumCells}\) cells vertically.

The degree of warping within each cell is defined by the values in \(x\text{WarpPos}\) and \(y\text{WarpPos}\) parameters. Each of these parameters must contain \((x\text{NumCells} + 1)\times(y\text{NumCells} + 1)\) values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose \(x\text{NumCells}\) is equal to 2 and \(y\text{NumCells}\) is equal to 1. Then the order of the data in the \(x\text{WarpPos}\) would be:

\[
x00, x10, x20, x01, x11, x21
\]

and in the \(y\text{WarpPos}\):

\[
y00, y10, y20, y01, y11, y21
\]

for a total of \((2 + 1)\times(1 + 1) = 6\) elements in each table.

**Parameters**

The function takes the following arguments:

- \(\text{dst}\) Pointer to destination image.
- \(\text{src}\) Pointer to source image.
xWarpPos  A float array of length \((x\text{NumCells} + 1) \times (y\text{NumCells} + 1)\) containing horizontal warp positions at the grid points, in row-major order.

yWarpPos  A float array of length \((x\text{NumCells} + 1) \times (y\text{NumCells} + 1)\) containing vertical warp positions at the grid points, in row-major order.

postShiftX  The displacement to apply to source X positions.

postShiftY  The displacement to apply to source Y positions.

xStart  The minimum X coordinate of the grid.

xStep  The horizontal spacing between grid cells.

xNumCells  The number of grid cell columns.

yStart  The minimum Y coordinate of the grid.

yStep  The vertical spacing between grid cells.

yNumCells  The number of grid cell rows.

filter  Type of resampling filter. It can be one of the following:

- MLIB_NEAREST
- MLIB_BILINEAR
- MLIB_BICUBIC
- MLIB_BICUBIC2

edge  Type of edge condition. It can be one of the following:

- MLIB_EDGE_DST_NO_WRITE
- MLIB_EDGE_SRC_PADDDED

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageGridWarp(3MLIB), mlib_ImageGridWarpTable(3MLIB), mlib_ImageGridWarpTable_Fp(3MLIB), attributes(5)
The `mlib_ImageGridWarpTable()` function performs a regular grid-based image warp with table-driven interpolation. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image; then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at (x0, y0), (x1, y0), (x0, y1) and (x1, y1), with source coordinates defined at each respective corner equal to (sx0, sy0), (sx1, sy1), (sx2, sy2) and (sx3, sy3), the source position (sx, sy) that maps onto (x, y) is given by the formulas:

\[
\begin{align*}
\text{xfrac} &= \frac{x - x0}{x1 - x0} \\
\text{yfrac} &= \frac{y - y0}{y1 - y0} \\
\text{s} &= \text{sx0} + (\text{sx1} - \text{sx0}) \times \text{xfrac} \\
\text{t} &= \text{sy0} + (\text{sy1} - \text{sy0}) \times \text{xfrac} \\
\text{u} &= \text{sx2} + (\text{sx3} - \text{sx2}) \times \text{xfrac} \\
\text{v} &= \text{sy2} + (\text{sy3} - \text{sy2}) \times \text{xfrac} \\
\text{sx} &= \text{s} + (\text{u} - \text{s}) \times \text{yfrac} - \text{postShiftX} \\
\text{sy} &= \text{t} + (\text{v} - \text{t}) \times \text{yfrac} - \text{postShiftY}
\end{align*}
\]

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically:
The results of above interpolation are shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at (xStart, yStart). Each cell has width equal to xStep and height equal to yStep, and there are xNumCells cells horizontally and yNumCells cells vertically.

The degree of warping within each cell is defined by the values in xWarpPos and yWarpPos parameters. Each of these parameters must contain (xNumCells + 1)*(yNumCells + 1) values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose xNumCells is equal to 2 and yNumCells is equal to 1. Then the order of the data in the xWarpPos would be:

\[ x00, x10, x20, x01, x11, x21 \]

and in the yWarpPos:

\[ y00, y10, y20, y01, y11, y21 \]

for a total of \((2 + 1)*(1 + 1) = 6\) elements in each table.

**Parameters**

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
**xWarpPos**  A float array of length \((x\text{NumCells} + 1)\times(y\text{NumCells} + 1)\) containing horizontal warp positions at the grid points, in row-major order.

**yWarpPos**  A float array of length \((x\text{NumCells} + 1)\times(y\text{NumCells} + 1)\) containing vertical warp positions at the grid points, in row-major order.

**postShiftX**  The displacement to apply to source X positions.

**postShiftY**  The displacement to apply to source Y positions.

**xStart**  The minimum X coordinate of the grid.

**xStep**  The horizontal spacing between grid cells.

**xNumCells**  The number of grid cell columns.

**yStart**  The minimum Y coordinate of the grid.

**yStep**  The vertical spacing between grid cells.

**yNumCells**  The number of grid cell rows.

**interp_table**  Pointer to an interpolation table. The table is created by the `mlib_ImageInterpTableCreate()` function.

**edge**  Type of edge condition. It can be one of the following:

- `MLIB_EDGE_DST_NO_WRITE`
- `MLIB_EDGE_SRC_PADDED`

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>attribute_type</th>
<th>attribute_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageInterpTableCreate(3MLIB)`, `mlib_ImageInterpTableDelete(3MLIB)`
- `mlib_ImageGridWarp(3MLIB)`, `mlib_ImageGridWarp_Fp(3MLIB)`
- `mlib_ImageGridWarpTable_Fp(3MLIB)`, attributes(5)
multimedia library functions - part 3

mlib_ImageGridWarpTable_Fp(3MLIB)

Name  mlib_ImageGridWarpTable_Fp — grid-based image warp with table-driven interpolation

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageGridWarpTable_Fp(mlib_image *dst,  
   const mlib_image *src, const mlib_f32 *xWarpPos,  
   const mlib_f32 *yWarpPos, mlib_d64 postShiftX, mlib_d64 postShiftY,  
   mlib_s32 xStart, mlib_s32 xStep, mlib_s32 xNumCells,  
   mlib_s32 yStart, mlib_s32 yStep, mlib_s32 yNumCells,  
   const void *interp_table, mlib_edge edge);

Description  The mlib_ImageGridWarpTable_Fp() function performs a regular grid-based image warp on a floating-point image with table-driven interpolation. The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be MLIB_FLOAT or MLIB_DOUBLE. The two images may have different sizes.

The image pixels are assumed to be centered at .5 coordinate points. For example, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point D is, first, backward mapped to a point S in the source image; then the source pixels with their centers surrounding point S are selected to do one of the interpolations specified by the filter parameter to generate the pixel value for point D.

The mapping from destination pixels to source positions is described by bilinear interpolation between a rectilinear grid of points with known mappings.

Given a destination pixel coordinate (x, y) that lies within a cell having corners at (x0, y0), (x1, y0), (x0, y1) and (x1, y1), with source coordinates defined at each respective corner equal to (sx0, sy0), (sx1, sy1), (sx2, sy2) and (sx3, sy3), the source position (sx, sy) that maps onto (x, y) is given by the formulas:

\[
\begin{align*}
\text{xfrac} & = \frac{x - x0}{x1 - x0} \\
\text{yfrac} & = \frac{y - y0}{y1 - y0} \\
\text{s} & = \text{sx0} + (\text{sx1} - \text{sx0})\times\text{xfrac} \\
\text{t} & = \text{sy0} + (\text{sy1} - \text{sy0})\times\text{xfrac} \\
\text{u} & = \text{sx2} + (\text{sx3} - \text{sx2})\times\text{xfrac} \\
\text{v} & = \text{sy2} + (\text{sy3} - \text{sy2})\times\text{xfrac} \\
\text{sx} & = \text{s} + (\text{u} - \text{s})\times\text{yfrac} - \text{postShiftX} \\
\text{sy} & = \text{t} + (\text{v} - \text{t})\times\text{yfrac} - \text{postShiftY}
\end{align*}
\]

In other words, the source x and y values are interpolated horizontally along the top and bottom edges of the grid cell, and the results are interpolated vertically.
The results of above interpolation are shifted by (-postShiftX, -postShiftY) to produce the source pixel coordinates.

The destination pixels that lie outside of any grid cells are kept intact. The grid is defined by a set of equal-sized cells. The grid starts at (xStart, yStart). Each cell has width equal to xStep and height equal to yStep, and there are xNumCells cells horizontally and yNumCells cells vertically.

The degree of warping within each cell is defined by the values in xWarpPos and yWarpPos parameters. Each of these parameters must contain (xNumCells + 1)*(yNumCells + 1) values, which, respectively, contain the source X and source Y coordinates that map to the upper-left corner of each cell in the destination image. The cells are enumerated in row-major order. That is, all the grid points along a row are enumerated first, then the grid points for the next row are enumerated, and so on.

For example, suppose xNumCells is equal to 2 and yNumCells is equal to 1. Then the order of the data in the xWarpPos would be:

x00, x10, x20, x01, x11, x21

and in the yWarpPos:

y00, y10, y20, y01, y11, y21

for a total of (2 + 1)*(1 + 1) = 6 elements in each table.

**Parameters**  The function takes the following arguments:

- **dest**  Pointer to destination image.
- **src**  Pointer to source image.
xWarpPos  A float array of length (xNumCells + 1)*(yNumCells + 1) containing horizontal warp positions at the grid points, in row-major order.
yWarpPos  A float array of length (xNumCells + 1)*(yNumCells + 1) containing vertical warp positions at the grid points, in row-major order.
postShiftX The displacement to apply to source X positions.
postShiftY The displacement to apply to source Y positions.
xStart The minimum X coordinate of the grid.
xStep The horizontal spacing between grid cells.
xNumCells The number of grid cell columns.
yStart The minimum Y coordinate of the grid.
yStep The vertical spacing between grid cells.
yNumCells The number of grid cell rows.
interp_table Pointer to an interpolation table. The table is created by the mlib_ImageInterpTableCreate() function.
edge Type of edge condition. It can be one of the following:
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_SRC_PADDED

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageInterpTableCreate(3MLIB), mlib_ImageInterpTableDelete(3MLIB), mlib_ImageGridWarp(3MLIB), mlib_ImageGridWarp_Fp(3MLIB), mlib_ImageGridWarpTable(3MLIB), attributes(5)
The `mlib_ImageHistogram2()` function creates a histogram by scanning an image, counting the number of pixels within a given range for each channel of the image, and then generating a histogram.

The image can have 1, 2, 3 or 4 channels. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The histogram must have the same number of channels as the image has.

One entry of the histogram, or a bin, is used to accumulate the number of pixels within a certain sub-range. The legal pixel range and the number of bins may be controlled separately.

If `binWidth` is defined as `(highValue - lowValue )/numBins` then bin i counts pixel values in the following range:

\[
\text{lowValue} + i*\text{binWidth} \leq x < \text{lowValue} + (i + 1)*\text{binWidth}
\]

The set of pixels scanned may furthermore be reduced by specifying `xPeriod` and `yPeriod` parameters that specify the sampling rate along each axis.

The set of pixels to be accumulated may be obtained from the following equations:

\[
x = x\text{Start} + p*x\text{Period}; \quad 0 \leq p < (w - x\text{Start})/x\text{Period}
\]

\[
y = y\text{Start} + q*y\text{Period}; \quad 0 \leq q < (h - y\text{Start})/y\text{Period}
\]

It is the user's responsibility to clear the histogram table before this function is called and to ensure that the histogram table supplied is suitable for the source image and the parameters. Otherwise, the result of this function is undefined.

The range from `lowValue[k]` to `(highValue[k] - 1)` must be a valid subrange of the image type range.

**Parameters**

- `histo` Pointer to histogram. The format of the histogram is `histo[\text{channel}][\text{index}]`. The index values for channel i can be \(0, 1, \ldots, \text{numBins}[i]-1\).
- `img` Pointer to source image.
- `numBins` The number of bins for each channel of the image.
- `lowValue` The lowest pixel value checked for each channel.
highValue  The highest pixel value checked for each channel. When counting the pixel values, highValue is not included.

xStart  The initial X sample coordinate.

yStart  The initial Y sample coordinate.

xPeriod  The X sampling rate. xPeriod ≥ 1.

yPeriod  The Y sampling rate. yPeriod ≥ 1.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageHistogram(3MLIB), attributes(5)
The `mlib_ImageHistogram()` function creates a histogram. The data type of the image can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:

- **histo**: Pointer to histogram. The format of the histogram is `histo[channel][index]`. The `MLIB_BYTE` type entries are indexed from 0 to 255. The `MLIB_SHORT` type entries are indexed from -32768 to -1, then from 0 to 32767. The `MLIB_USHORT` type entries are indexed from 0 to 65535. The `MLIB_INT` type entries are indexed from -2147483648 to -1, then from 0 to 2147483647.

- **img**: Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageHistogram2(3MLIB), attributes(5)`
### Name
mlib_ImageInterpTableCreate – creates an interpolation table

### Synopsis
```c
#include <mlib.h>

void *mlib_ImageInterpTableCreate(mlib_type type, mlib_s32 width,
                                 mlib_s32 height, mlib_s32 leftPadding, mlib_s32 topPadding,
                                 mlib_s32 subsampleBitsH, mlib_s32 subsampleBitsV,
                                 mlib_s32 precisionBits, const void *dataH, const void *dataV);
```

### Description
The `mlib_ImageInterpTableCreate()` function creates an interpolation table based on parameters specified.

This function creates an internal data structure, an interpolation table, which can be used by some image geometric functions for implementing a table-driven interpolation algorithm.

The parameter type defines the type of `dataH/dataV` input arrays and can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

The `dataH` array should have at least `width*2**subsampleBitsH` entries. `dataH[i*2**subsampleBitsH]` holds the coefficient for the leftmost neighboring pixel, `dataH[i*2**subsampleBitsH + 1]` holds the coefficient for the second neighboring pixel from left, ..., and `dataH[i*2**subsampleBitsH + width - 1]` holds the coefficient for the rightmost neighboring pixel, where `i = 0, 1, 2, ..., 2**subsampleBitsH - 1`.

The `dataV` array should have at least `height*2**subsampleBitsV` entries or should be NULL. If `dataV` is NULL, then `dataH` is used in its place, and in this case the parameters `topPadding`, `height`, and `subsampleBitsV` are ignored.

### Parameters
The function takes the following arguments:

- **type**
  Data type of the coefficients.

- **width**
  Width of the interpolation kernel in pixels.

- **height**
  Height of the interpolation kernel in pixels.

- **leftPadding**
  Number of pixels lying to the left of the interpolation kernel key position.

- **topPadding**
  Number of pixels lying above the interpolation kernel key position.

- **subsampleBitsH**
  Numbers of bits used for the horizontal subsample position.

- **subsampleBitsV**
  Numbers of bits used for the vertical subsample position.

- **precisionBits**
  Number of fractional bits used to describe the coefficients.

- **dataH**
  Pointer to horizontal coefficient data.

- **dataV**
  Pointer to vertical coefficient data.
The function returns a pointer to an interpolation table.

**Return Values**  
The function returns a pointer to an interpolation table.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageInterpTableDelete(3MLIB), mlib_ImageAffineTable(3MLIB),  
mlib_ImageZoomTranslateTable(3MLIB), mlib_ImageGridWarpTable(3MLIB),  
mlib_ImagePolynomialWarpTable(3MLIB), attributes(5)
mlib_ImageInterpTableDelete – deletes an interpolation table

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_ImageInterpTableDelete(void *interp_table);

Description

The mlib_ImageInterpTableDelete() function deletes an interpolation table.

This function deletes the structure of an interpolation table and frees the memory allocated by mlib_ImageInterpTableCreate().

Parameters

The function takes the following arguments:

interp_table Pointer to an interpolation table.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageInterpTableCreate(3MLIB), mlib_ImageAffineTable(3MLIB),
mlib_ImageZoomTranslateTable(3MLIB), mlib_ImageGridWarpTable(3MLIB),
mlib_ImagePolynomialWarpTable(3MLIB), attributes(5)
The `mlib_ImageInvert()` function performs the inversion of an image such that white becomes black, light gray becomes dark gray, and so on. It uses the following equation:

\[ \text{dst}[x][y][i] = (G_{\text{white}} + G_{\text{black}}) - \text{src}[x][y][i] \]

The values of \(G_{\text{white}}\) and \(G_{\text{black}}\) for different types of images are:

<table>
<thead>
<tr>
<th>Image Type</th>
<th>(G_{\text{white}})</th>
<th>(G_{\text{black}})</th>
<th>(G_{\text{white}} + G_{\text{black}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BYTE</td>
<td>255</td>
<td>0</td>
<td>255 (0xFF)</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>32767</td>
<td>-32768</td>
<td>-1 (0xFFFFFFFF)</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>65535</td>
<td>0</td>
<td>65535 (0xFFFF)</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>2147483647</td>
<td>-2147483648</td>
<td>-1 (0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

Given that integer data are in the two's complement representation, `mlib_ImageInvert()` is the same as `mlib_ImageNot()`, while `mlib_ImageInvert_Inp()` is the same as `mlib_ImageNot_Inp()`.

**Parameters**
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageInvert_Inp(3MLIB), mlib_ImageInvert_Fp(3MLIB), mlib_ImageInvert_Fp_Inp(3MLIB), attributes(5)`
The `mlib_ImageInvert_Fp()` function performs the floating-point inversion of an image such that white becomes black, light gray becomes dark gray, and so on. It uses the following equation:
\[
dst[x][y][i] = -src[x][y][i]
\]

The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageInvert_Fp_Inp(3MLIB), mlib_ImageInvert(3MLIB),
mlib_ImageInvert_Inp(3MLIB), attributes(5)`
mlib_ImageInvert_Fp_Inp(3MLIB)

Name     mlib_ImageInvert_Fp_Inp – invert

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>

          mlib_status mlib_ImageInvert_Fp_Inp(mlib_image *srcdst);

Description The mlib_ImageInvert_Fp_Inp() function performs the floating-point inversion of an image such that white becomes black, light gray becomes dark gray, and so on, in place.

It uses the following equation:

          srcdst[x][y][i] = -srcdst[x][y][i]

Parameters  The function takes the following arguments:

          srcdst    Pointer to source and destination image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageInvert_Fp(3MLIB), mlib_ImageInvert(3MLIB),
           mlib_ImageInvert_Inp(3MLIB), attributes(5)
# mlib_ImageInvert_Inp

## Synopsis
```
c { flag... } file... -l mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageInvert_Inp(mlib_image *srcdst);
```

## Description
The `mlib_ImageInvert_Inp()` function performs the in-place inversion of an image such that white becomes black, light gray becomes dark gray, and so on. It uses the following equation:

\[
srcdst[x][y][i] = (Gwhite + Gblack) - srcdst[x][y][i]
\]

The values of Gwhite and Gblack for different types of images are:

<table>
<thead>
<tr>
<th>Image Type</th>
<th>Gwhite</th>
<th>Gblack</th>
<th>Gwhite + Gblack</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BYTE</td>
<td>255</td>
<td>0</td>
<td>255 (0xFF)</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>32767</td>
<td>-32768</td>
<td>-1 (0xFFFF)</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>65535</td>
<td>0</td>
<td>65535 (0xFFFF)</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>2147483647</td>
<td>-2147483648</td>
<td>-1 (0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

Given that integer data are in the two’s complement representation, `mlib_ImageInvert()` is the same as `mlib_ImageNot()`, while `mlib_ImageInvert_Inp()` is the same as `mlib_ImageNot_Inp()`.

## Parameters
The function takes the following arguments:
- `srcdst` Pointer to source and destination image.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageInvert(3MLIB)`
- `mlib_ImageInvert_Fp(3MLIB)`
- `mlib_ImageInvert_Fp_Inp(3MLIB)`
mlib_ImageIsNotAligned2

### Synopsis

```c
#include <mlib.h>

int mlib_ImageIsNotAligned2(const mlib_image *img);
```

### Description

The `mlib_ImageIsNotAligned2()` function tests for a specific alignment of a mediaLib image structure.

### Parameters

The function takes the following arguments:

- `img`  
  Pointer to source image.

### Return Values

Returns 0 if data address is two-byte aligned; otherwise, returns nonzero.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned8(3MLIB)`
- `mlib_ImageIsNotAligned64(3MLIB)`, `mlib_ImageIsNotHeight2X(3MLIB)`
- `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`
- `mlib_ImageIsNotOneDvector(3MLIB)`, `mlib_ImageIsNotStride8X(3MLIB)`
- `mlib_ImageIsNotWidth2X(3MLIB)`, `mlib_ImageIsNotWidth4X(3MLIB)`
- `mlib_ImageIsNotWidth8X(3MLIB)`, `mlib_ImageIsUserAllocated(3MLIB)`, `attributes(5)`
The `mlib_ImageIsNotAligned4()` function tests for a specific alignment of a mediaLib image structure.

**Parameters**
The function takes the following arguments:

- `img` Pointer to source image.

**Return Values**
Returns 0 if data address is four-byte aligned; otherwise, returns nonzero.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageIsNotAligned2(3MLIB), mlib_ImageIsNotAligned8(3MLIB), mlib_ImageIsNotAligned64(3MLIB), mlib_ImageIsNotHeight2X(3MLIB), mlib_ImageIsNotHeight4X(3MLIB), mlib_ImageIsNotHeight8X(3MLIB), mlib_ImageIsNotOneDvector(3MLIB), mlib_ImageIsNotStride8X(3MLIB), mlib_ImageIsNotWidth2X(3MLIB), mlib_ImageIsNotWidth4X(3MLIB), mlib_ImageIsNotWidth8X(3MLIB), mlib_ImageIsUserAllocated(3MLIB), attributes(5)"
The `mlib_ImageIsNotAligned64()` function tests for a specific alignment characteristic of a mediaLib image structure.

Parameters
The function takes the following arguments:
- `img` Pointer to source image.

Return Values
Returns 0 if data address is 64-byte aligned; otherwise, returns nonzero.

Attributes
See `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

See Also
- `mlib_ImageIsNotAligned2(3MLIB)`, `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned8(3MLIB)`, `mlib_ImageIsNotHeight2X(3MLIB)`, `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`, `mlib_ImageIsNotOneDvector(3MLIB)`, `mlib_ImageIsNotStride8X(3MLIB)`, `mlib_ImageIsNotWidth2X(3MLIB)`, `mlib_ImageIsNotWidth4X(3MLIB)`, `mlib_ImageIsNotWidth8X(3MLIB)`, `mlib_ImageIsUserAllocated(3MLIB)`
The `mlib_ImageIsNotAligned8()` function tests for a specific alignment of a mediaLib image structure.

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

int mlib_ImageIsNotAligned8(const mlib_image *img);
```

**Description**
The `mlib_ImageIsNotAligned8()` function takes the following arguments:

- `img`  
  Pointer to source image.

**Parameters**
Returns 0 if data address is eight-byte aligned; otherwise, returns nonzero.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageIsNotAligned2(3MLIB)`, `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned64(3MLIB)`, `mlib_ImageIsNotHeight2X(3MLIB)`, `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`, `mlib_ImageIsNotOneDvector(3MLIB)`, `mlib_ImageIsNotStride8X(3MLIB)`, `mlib_ImageIsNotWidth2X(3MLIB)`, `mlib_ImageIsNotWidth4X(3MLIB)`, `mlib_ImageIsNotWidth8X(3MLIB)`, `mlib_ImageIsUserAllocated(3MLIB)`, attributes(5)
mlib_ImageIsNotHeight2X(3MLIB)

Name  mlib_ImageIsNotHeight2X – image query, 2X height

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

int mlib_ImageIsNotHeight2X(const mlib_image *img);

Description  The mlib_ImageIsNotHeight2X() function tests for a specific height characteristic of a mediaLib image structure.

Parameters  The function takes the following arguments:

    img    Pointer to source image.

Return Values  Returns 0 if height is a multiple of two; otherwise, returns nonzero.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageIsNotAligned2(3MLIB), mlib_ImageIsNotAligned4(3MLIB), mlib_ImageIsNotAligned8(3MLIB), mlib_ImageIsNotAligned64(3MLIB), mlib_ImageIsNotHeight4X(3MLIB), mlib_ImageIsNotHeight8X(3MLIB), mlib_ImageIsNotOneDvector(3MLIB), mlib_ImageIsNotStride8X(3MLIB), mlib_ImageIsNotWidth2X(3MLIB), mlib_ImageIsNotWidth4X(3MLIB), mlib_ImageIsNotWidth8X(3MLIB), mlib_ImageIsUserAllocated(3MLIB), attributes(5)
mlib_ImageIsNotHeight4X(3MLIB)

Name mlib_ImageIsNotHeight4X – image query, 4X height

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

int mlib_ImageIsNotHeight4X(const mlib_image *img);

Description The mlib_ImageIsNotHeight4X() function tests for a specific height characteristic of a mediaLib image structure.

Parameters The function takes the following arguments:

  img  Pointer to source image.

Return Values Returns 0 if height is a multiple of four; otherwise, returns nonzero.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageIsNotAligned2(3MLIB), mlib_ImageIsNotAligned4(3MLIB),
mlib_ImageIsNotAligned8(3MLIB), mlib_ImageIsNotAligned64(3MLIB),
mlib_ImageIsNotHeight2X(3MLIB), mlib_ImageIsNotHeight8X(3MLIB),
mlib_ImageIsNotOneDvector(3MLIB), mlib_ImageIsNotStride8X(3MLIB),
mlib_ImageIsNotWidth2X(3MLIB), mlib_ImageIsNotWidth4X(3MLIB),
mlib_ImageIsNotWidth8X(3MLIB), mlib_ImageIsUserAllocated(3MLIB), attributes(5)
The `mlib_ImageIsNotHeight8X()` function tests for a specific height characteristic of a mediaLib image structure.

**Parameters**
The function takes the following arguments:

- `img` Pointer to source image.

**Return Values**
Returns 0 if height is a multiple of eight; otherwise, returns nonzero.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageIsNotAligned2(3MLIB), mlib_ImageIsNotAligned4(3MLIB), mlib_ImageIsNotAligned8(3MLIB), mlib_ImageIsNotAligned64(3MLIB), mlib_ImageIsNotHeight2X(3MLIB), mlib_ImageIsNotHeight4X(3MLIB), mlib_ImageIsNotOneDvector(3MLIB), mlib_ImageIsNotStride8X(3MLIB), mlib_ImageIsNotWidth2X(3MLIB), mlib_ImageIsNotWidth4X(3MLIB), mlib_ImageIsNotWidth8X(3MLIB), mlib_ImageIsUserAllocated(3MLIB), attributes(5)`
The `mlib_ImageIsNotOneDvector()` function tests for a specific dimension characteristic of a mediaLib image structure.

**Parameters**
The function takes the following arguments:

- `img` Pointer to source image.

**Return Values**
Returns 0 if data space can be treated as a 1D vector; otherwise, returns nonzero.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageIsNotAligned2(3MLIB), mlib_ImageIsNotAligned4(3MLIB), mlib_ImageIsNotAligned8(3MLIB), mlib_ImageIsNotAligned64(3MLIB), mlib_ImageIsNotHeight2X(3MLIB), mlib_ImageIsNotHeight4X(3MLIB), mlib_ImageIsNotHeight8X(3MLIB), mlib_ImageIsNotStride8X(3MLIB), mlib_ImageIsNotWidth2X(3MLIB), mlib_ImageIsNotWidth4X(3MLIB), mlib_ImageIsNotWidth8X(3MLIB), mlib_ImageIsUserAllocated(3MLIB), attributes(5)`
Name mlib_ImageIsNotStride8X – image query, 8X stride

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

    int mlib_ImageIsNotStride8X(const mlib_image *img);

Description The mlib_ImageIsNotStride8X() function tests for a specific stride characteristic of a
mediaLib image structure.

Parameters The function takes the following arguments:

    img Pointer to source image.

Return Values Returns 0 if stride is a multiple of eight; otherwise, returns nonzero.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageIsNotAligned2(3MLIB), mlib_ImageIsNotAligned4(3MLIB),
mlib_ImageIsNotAligned8(3MLIB), mlib_ImageIsNotAligned64(3MLIB),
mlib_ImageIsNotHeight2X(3MLIB), mlib_ImageIsNotHeight4X(3MLIB),
mlib_ImageIsNotHeight8X(3MLIB), mlib_ImageIsNotOneDvector(3MLIB),
mlib_ImageIsNotWidth2X(3MLIB), mlib_ImageIsNotWidth4X(3MLIB),
mlib_ImageIsNotWidth8X(3MLIB), mlib_ImageIsUserAllocated(3MLIB), attributes(5)
The `mlib_ImageIsNotWidth2X()` function tests for a specific width characteristic of a MediaLib image structure.

**Synopsis**

```c
cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

int mlib_ImageIsNotWidth2X(const mlib_image *img);
```

**Parameters**

The function takes the following arguments:

- `img` Pointer to source image.

**Return Values**

Returns 0 if width is a multiple of two; otherwise, returns nonzero.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageIsNotAligned2(3MLIB)`, `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned8(3MLIB)`, `mlib_ImageIsNotAligned64(3MLIB)`
- `mlib_ImageIsNotHeight2X(3MLIB)`, `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`
- `mlib_ImageIsNotStride8X(3MLIB)`, `mlib_ImageIsNotWidth4X(3MLIB)`
- `mlib_ImageIsUserAllocated(3MLIB)`, `attributes(5)`
The `mlib_ImageIsNotWidth4X()` function tests for a specific width characteristic of a mediaLib image structure.

**Parameters**
The function takes the following arguments:

- `img` Pointer to source image.

**Return Values**
Returns 0 if width is a multiple of four; otherwise, returns nonzero.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageIsNotAligned2(3MLIB)`, `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned8(3MLIB)`, `mlib_ImageIsNotAligned64(3MLIB)`, `mlib_ImageIsNotHeight2X(3MLIB)`, `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`, `mlib_ImageIsNotOneDvector(3MLIB)`, `mlib_ImageIsNotStride8X(3MLIB)`, `mlib_ImageIsNotWidth2X(3MLIB)`, `mlib_ImageIsNotWidth8X(3MLIB)`, `mlib_ImageIsUserAllocated(3MLIB)`, attributes(5)
The `mlib_ImageIsNotWidth8X()` function tests for a specific width characteristic of a `mediaLib` image structure.

**Parameters**
The function takes the following arguments:

- `img` Pointer to source image.

**Return Values**
Returns 0 if width is a multiple of eight; otherwise, returns nonzero.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageIsNotAligned2(3MLIB)`, `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned8(3MLIB)`, `mlib_ImageIsNotAligned64(3MLIB)`
- `mlib_ImageIsNotHeight2X(3MLIB)`, `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`
- `mlib_ImageIsNotStride2X(3MLIB)`, `mlib_ImageIsNotStride4X(3MLIB)`
- `mlib_ImageIsNotWidth2X(3MLIB)`, `mlib_ImageIsNotWidth4X(3MLIB)`
- `mlib_ImageIsUserAllocated(3MLIB)`, `attributes(5)`
The `mlib_ImageIsUserAllocated()` function tests for a specific allocation characteristic of a mediaLib image structure. The function takes the following arguments:

- `src` Pointer to source image.

Returns 0 if data space has been allocated by mediaLib; otherwise, returns nonzero.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageIsNotAligned2(3MLIB)`, `mlib_ImageIsNotAligned4(3MLIB)`, `mlib_ImageIsNotAligned8(3MLIB)`, `mlib_ImageIsNotAligned64(3MLIB)`, `mlib_ImageIsNotHeight2X(3MLIB)`, `mlib_ImageIsNotHeight4X(3MLIB)`, `mlib_ImageIsNotHeight8X(3MLIB)`, `mlib_ImageIsNotOneDvector(3MLIB)`, `mlib_ImageIsNotStride8X(3MLIB)`, `mlib_ImageIsNotWidth2X(3MLIB)`, `mlib_ImageIsNotWidth4X(3MLIB)`, `mlib_ImageIsNotWidth8X(3MLIB)`, attributes(5)
mlib_ImageLog – computes the natural logarithm of the image pixels.

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageLog(mlib_image *dst, const mlib_image *src);
```

Description

The `mlib_ImageLog()` function computes the natural logarithm of the image pixels. It uses the following equation:

\[
\text{dst}[x][y][i] = \log(\text{src}[x][y][i])
\]

Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

See Also

`mlib_ImageLog_Fp(3MLIB), mlib_ImageLog_Fp_Inp(3MLIB), mlib_ImageLog_Inp(3MLIB), attributes(5)`
The `mlib_ImageLog_Fp()` function computes the natural logarithm of the image pixels. It uses the following equation:

$$\text{dst}[x][y][i] = \log(\text{src}[x][y][i])$$

The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageLog(3MLIB), mlib_ImageLog_Fp_Inp(3MLIB), mlib_ImageLog_Inp(3MLIB), attributes(5)`
The `mlib_ImageLog_Fp_Inp()` function computes the natural logarithm of the image pixels, in place.

It uses the following equation:

\[
srcest[x][y][i] = \log(srcest[x][y][i])
\]

**Parameters**

- `srcdst` Pointer to source and destination image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageLog(3MLIB)`, `mlib_ImageLog_Fp(3MLIB)`, `mlib_ImageLog_Inp(3MLIB)`, attributes(5)
mlib_ImageLog_Inp(3MLIB)

**Name**
mlib_ImageLog_Inp – computes the natural logarithm of the image pixels, in place

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageLog_Inp(mlib_image *srcdst);
```

**Description**
The `mlib_ImageLog_Inp()` function computes the natural logarithm of the image pixels in place.

It uses the following equation:
```
srcdst[x][y][i] = log(srcdst[x][y][i])
```

**Parameters**
The function takes the following arguments:
- `srcdst` Pointer to source and destination image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageLog(3MLIB)`
- `mlib_ImageLog_Fp(3MLIB)`
- `mlib_ImageLog_Fp_Inp(3MLIB)`
- attributes(5)
mlib_ImageLookUp2 – table lookup

#include <mlib.h>

mlib_status mlib_ImageLookUp2(mlib_image *dst, const mlib_image *src, const void **table, const mlib_s32 *offsets, mlib_s32 channels);

Description
The mlib_ImageLookUp2() function maps the source image to the destination image by using the user-specified lookup table and an offset.

The source and destination images must have the same width and height.

The source and destination images can have different data types. See the following table for available variations of the table lookup function on image types:

<table>
<thead>
<tr>
<th>Type[*]</th>
<th>BYTE</th>
<th>SHORT</th>
<th>USHORT</th>
<th>INT</th>
<th>FLOAT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BIT</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[* Each row represents a source data type. Each column represents a destination data type.

The source and destination images also can have a different number of channels. The source image can be a single-channel image or can have the same number of channels as the destination image. The lookup table can have one channel or have the same channels as the destination image. See the following table for possible variations on the number of channels in the images and the lookup table:

<table>
<thead>
<tr>
<th># of channels in the input image</th>
<th># of channels in the lookup table</th>
<th># of channels in the output image</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>l</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

where, n = 1, 2, 3, 4.

Each of the following equations is used in the corresponding case shown in the table above.
dst[x][y][i] = table[i][src[x][y][0] - offsets[i]]
dst[x][y][i] = table[0][src[x][y][i] - offsets[0]]
dst[x][y][i] = table[i][src[x][y][i] - offsets[i]]

Parameters
The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
table Pointer to lookup table. The data type of the lookup table is the same as that of the destination image. The format of the lookup table is:

\[
table[channel][index]
\]

The entries are indexed from 0 to 1, 2, ..., and so on. It is the user’s responsibility to provide a lookup table that has enough entries to cover all possible values of the pixel components deducted by the offset in each channel of the source image.

offsets Offset values subtracted from the src pixel before table lookup.

channels Number of channels in the lookup table. If the number of channels equals 1, then the same table is applied to all channels. Otherwise, the number of channels must be no less than the number of channels in the destination image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageLookUp(3MLIB), mlib_ImageLookUp_Inp(3MLIB), mlib_ImageLookUpMask(3MLIB), attributes(5)
The `mlib_ImageLookUp()` function maps the source image to the destination image by using the user-specified lookup table.

The source and destination images must have the same width and height. The source image can be a single channel image or can have the same number of channels as the destination image. One of the following equations is used accordingly:

```
        dst[x][y][i] = table[i][src[x][y][0]]
        dst[x][y][i] = table[i][src[x][y][i]]
```

The source and destination images can have different data types. See the following table for available variations of the table lookup function on image types:

<table>
<thead>
<tr>
<th>Type[*]</th>
<th>BYTE</th>
<th>SHORT</th>
<th>USHORT</th>
<th>INT</th>
<th>FLOAT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BIT</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[∗] Each row represents a source data type. Each column represents a destination data type.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `table` Pointer to lookup table. The data type of the lookup table is the same as the destination image. The number of entries in the lookup table is determined by the type of the input image. The format of the lookup table is:

  `table[channel][index]`

  The MLIB_BIT type entries are indexed from 0 to 1. The MLIB_BYTE type entries are indexed from 0 to 255. The MLIB_SHORT type entries are indexed from -32768 to -1,
then from 0 to 32767. The MLIB_USHORT type entries are indexed from 0 to 65535. The MLIB_INT type entries are indexed from -2147483648 to -1, and then from 0 to 2147483647.

If a table covering the full range of input data type is not available or not realistic, which is mostly true for doing table lookup with an MLIB_INT input image, a smaller table can be used. In this case, the pointer to the table has to be adjusted as if it is pointing to the element for the smallest value of the input data type. For example, to use a table covering input data range of [-65536, 65535], the pointer needs to be adjusted as follows:

```
table_16_32[0] += MLIB_S32_MIN + 65536;
```

This might cause a pointer arithmetic overflow in 32-bit mode, but probably works if the overflow is handled as a wrap-around. If possible, function `mlib_ImageLookUp2()` should be used instead.

**Return Values** The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_ImageLookUp_Inp(3MLIB)`, `mlib_ImageLookUp2(3MLIB)`, `mlib_ImageLookUpMask(3MLIB)`, attributes(5)
The `mlib_ImageLookUp_Inp()` function maps the source image to the destination image, in place, by using the user-specified lookup table.

The following equation is used:

\[
\text{srcdst}[x][y][i] = \text{table}[i][\text{srcdst}[x][y][i]]
\]

The function takes the following arguments:

- `srcdst` Pointer to first source and destination image.
- `table` Pointer to lookup table. The data type of the lookup table is the same as the destination image. The number of entries in the lookup table is determined by the type of the input image. The format of the lookup table is:

  \[
  \text{table}[\text{channel}][\text{index}]
  \]

  The `MLIB_BYTE` type entries are indexed from 0 to 255. The `MLIB_SHORT` type entries are indexed from -32768 to 0, then from 0 to 32767. The `MLIB_USHORT` type entries are indexed from 0 to 65535. The `MLIB_INT` type entries are indexed from -2147483648 to -1, and then from 0 to 2147483647.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageLookUp(3MLIB), mlib_ImageLookUp2(3MLIB), mlib_ImageLookUpMask(3MLIB), attributes(5)`
The `mlib_ImageLookUpMask()` function maps the source image to the destination image by using the user-specified lookup table and applying a channel mask.

The source and destination images must have the same width and height. The source image can be a single channel image or can have the same number of channels as the destination image. One of the following equations is used accordingly:

```
dst[x][y][i] = table[i][src[x][y][0]]
dst[x][y][i] = table[i][src[x][y][i]]
```

The source and destination images can have different data types. See the following table for available variations of the table lookup function on image types:

<table>
<thead>
<tr>
<th>Type[*]</th>
<th>BYTE</th>
<th>SHORT</th>
<th>USHORT</th>
<th>INT</th>
<th>FLOAT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BIT</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[*] Each row represents a source data type. Each column represents a destination data type.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `table` Pointer to lookup table. The data type of the lookup table is the same as the destination image. The number of entries in the lookup table is determined by the type of the input image. The format of the lookup table is:

  ```
table[channel][index]
  ```

The MLIB_BIT type entries are indexed from 0 to 1. The MLIB_BYTE type entries are indexed from 0 to 255. The MLIB_SHORT type entries are indexed from -32768
to -1, then from 0 to 32767. The MLIB_USHORT type entries are indexed from 0 to 65535. The MLIB_INT type entries are indexed from -2147483648 to -1, and then from 0 to 2147483647.

channels  Number of channels in the lookup table. If the number of channels is equal to 1, then the same table is applied to all channels. Otherwise, the number of channels must be no less than the number of valid 1s in the channel mask.

cmask Channel mask. Each bit of the mask represents a channel of an image or a lookup table. Only the rightmost four bits of cmask are considered, where the least significant bit of cmask is for the last channel. The channels corresponding to 0 bits of cmask are not processed or used. cmask is always applied to the destination image dst. If the source image src has the same number of channels as dst, then cmask is also applied to src. Otherwise, each channel of src is used for each cmask bit with a value of 1, in this order: the first channel for the first 1 from the left in cmask. If src has only one channel, then the same src channel is used for every cmask bit with a value of 1. If the lookup table has the same number of channels as dst, then cmask is also applied to table. Otherwise, each table channel is used for each cmask bit with a value of 1, in this order: the first channel for the first 1 from the left in cmask. If table has only one channel, then the same table channel is used for every cmask bit with a value of 1.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageLookUp(3MLIB), mlib_ImageLookUp_Inp(3MLIB), mlib_ImageLookUp2(3MLIB), attributes(5)
The `mlib_ImageMax()` function accepts input from the two source images and writes the maximum to the destination image on a pixel-by-pixel basis. It uses the following equation:

\[
dst[x][y][i] = \text{MAX} \{ \text{src1}[x][y][i], \text{src2}[x][y][i] \}
\]

**Parameters**

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageMax_Fp(3MLIB), mlib_ImageMax_Fp_Inp(3MLIB), mlib_ImageMax_Inp(3MLIB), attributes(5)
mlib_ImageMaxFilter3x3 – 3x3 maximum filter

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMaxFilter3x3(mlib_image *dst, const mlib_image *src);

Description

The mlib_ImageMaxFilter3x3() function replaces the center pixel in a neighborhood with the maximum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
\text{dst}[x][y][0] = \text{MAX} \{ \text{src}[p][q][0], x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}
\]

where \(x = 1, \ldots, w - 2\); \(y = 1, \ldots, h - 2\).

Parameters

The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageMaxFilter3x3_Fp(3MLIB), mlib_ImageMaxFilter5x5(3MLIB),
mlib_ImageMaxFilter5x5_Fp(3MLIB), mlib_ImageMaxFilter7x7(3MLIB),
mlib_ImageMaxFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter3x3(3MLIB),
mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter3x3_US(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMaxFilter3x3_Fp()` function replaces the center pixel in a neighborhood with the floating-point maximum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
\text{dst}[x][y][0] = \text{MAX}\{ \text{src}[p][q][0], \ x-1 \leq p \leq x+1; y-1 \leq q \leq y+1 \}
\]

where \(x = 1, \ldots, w - 2; y = 1, \ldots, h - 2\).

**Parameters**

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter5x5(3MLIB)`,
- `mlib_ImageMaxFilter7x7(3MLIB)`,
- `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3_US(3MLIB)`,
- `mlib_ImageMedianFilter5x5(3MLIB)`, `mlib_ImageMedianFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMedianFilter5x5_US(3MLIB)`,
- `mlib_ImageMedianFilter7x7(3MLIB)`,
- `mlib_ImageMedianFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter7x7_US(3MLIB)`,
- `mlib_ImageMedianFilterMxN(3MLIB)`,
- `mlib_ImageMedianFilterMxN_Fp(3MLIB)`,
- `mlib_ImageMedianFilterMxN_US(3MLIB)`,
mlib_ImageMaxFilter3x3_Fp(3MLIB)

mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
Name  
mlib_ImageMaxFilter5x5 — 5x5 Max Filter

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMaxFilter5x5(mlib_image *dst, const mlib_image *src);

Description  
The mlib_ImageMaxFilter5x5() function replaces the center pixel in a neighborhood with the maximum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
dst[x][y][0] = \max\{ src[p][q][0], \quad x-2 \leq p \leq x+2; \quad y-2 \leq q \leq y+2 \}
\]

where \(x = 2, \ldots, w - 3; \quad y = 2, \ldots, h - 3\).

Parameters  
The function takes the following arguments:

- \(dst\) Pointer to destination image.
- \(src\) Pointer to source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5_Fp(3MLIB), mlib_ImageMaxFilter7x7(3MLIB),
mlib_ImageMaxFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter3x3(3MLIB),
mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter3x3_US(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB)

mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageMaxFilter5x5_Fp - 5x5 Max Filter

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMaxFilter5x5_Fp(mlib_image *dst,
const mlib_image *src);

Description

The mlib_ImageMaxFilter5x5_Fp() function replaces the center pixel in a neighborhood with the floating-point maximum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

dst[x][y][0] = MAX{ src[p][q][0],
x-2 ≤ p ≤ x+2; y-2 ≤ q ≤ y+2 }

where x = 2, ..., w - 3; y = 2, ..., h - 3.

Parameters

The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB),
mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageMaxFilter5x5_Fp(3MLIB)

mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
Name
mlib_ImageMaxFilter7x7 – 7x7 Max Filter

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMaxFilter7x7(mlib_image *dst, const mlib_image *src);

Description
The mlib_ImageMaxFilter7x7() function replaces the center pixel in a neighborhood with the maximum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

dst[x][y][0] = MAX{ src[p][q][0],
                      x-3 ≤ p ≤ x+3; y-3 ≤ q ≤ y+3 }  

where x = 3, ..., w - 4; y = 3, ..., h - 4.

Parameters
The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB), mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB), mlib_ImageRankFilterMxN_US(3MLIB), mlib_ImageMinFilterMxN_US(3MLIB), mlib_ImageMinFilterMxN_Fp(3MLIB), mlib_ImageMinFilterMxN_US_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB)

mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB), mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)


Name  mlib_ImageMaxFilter7x7_Fp – 7x7 Max Filter

Synopsis  

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMaxFilter7x7_Fp(mlib_image *dst, const mlib_image *src);

Description  
The mlib_ImageMaxFilter7x7_Fp() function replaces the center pixel in a neighborhood with the floating-point maximum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

dst[x][y][0] = MAX{ src[p][q][0],
                   x-3 ≤ p ≤ x+3; y-3 ≤ q ≤ y+3 }

where x = 3, ..., w - 4; y = 3, ..., h - 4.

Parameters  
The function takes the following arguments:

    dst Pointer to destination image.
    src Pointer to source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageMaxFilter7x7_Fp(3MLIB)

mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMax_Fp()` function accepts input from the two floating-point source images and writes the maximum to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = \max\{ \text{src1}[x][y][i], \text{src2}[x][y][i] \}
\]

### Parameters
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
`mlib_ImageMax(3MLIB), mlib_ImageMax_Fp_Inp(3MLIB), mlib_ImageMax_Inp(3MLIB), attributes(5)"
The `mlib_ImageMax_Fp_Inp()` function accepts input from the two floating-point source images and writes the maximum, in place, on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{src1dst}[x][y][i] = \text{MAX} \{ \text{src1dst}[x][y][i], \text{src2}[x][y][i] \} \]

**Parameters**

- `src1dst` : Pointer to source and destination image.
- `src2` : Pointer to second source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageMax(3MLIB), mlib_ImageMax_Fp(3MLIB), mlib_ImageMax_Inp(3MLIB), attributes(5)`
mlib_ImageMaximum (3MLIB)

Name
mlib_ImageMaximum – image maximum

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMaximum(mlib_s32 *max, const mlib_image *img);

Description
The mlib_ImageMaximum() function determines the maximum value for each channel in an image.

It uses the following equation:

\[ \max[i] = \max(\text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h) \]

Parameters
The function takes the following arguments:

- \( \text{max} \) – Pointer to maximum vector, where length is the number of channels in the image. \( \max[i] \) contains the maximum of channel \( i \).
- \( \text{img} \) – Pointer to a source image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageMaximum_Fp(3MLIB), mlib_ImageMinimum(3MLIB), mlib_ImageMinimum_Fp(3MLIB), attributes(5)
The `mlib_ImageMaximum_Fp()` function determines the maximum value for each channel in a floating-point image.

It uses the following equation:

\[
\text{max}[i] = \max \{ \text{img}[x][y][i] ; \ 0 \leq x < w, \ 0 \leq y < h \}
\]

The function takes the following arguments:

- `max`   Pointer to maximum vector, where length is the number of channels in the image.
- `img`   Pointer to a source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageMaximum(3MLIB), mlib_ImageMinimum(3MLIB), mlib_ImageMinimum_Fp(3MLIB), attributes(5)`
The `mlib_ImageMax_Inp()` function accepts input from the two source images and writes the maximum in place on a pixel-by-pixel basis. It uses the following equation:

\[
\text{src1dst}[x][y][i] = \text{MAX} \{ \text{src1dst}[x][y][i], \text{src2}[x][y][i] \}
\]

**Parameters**

- `src1dst` Pointer to source and destination image.
- `src2` Pointer to second source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageMax(3MLIB), mlib_ImageMax_Fp(3MLIB), mlib_ImageMax_Fp_Inp(3MLIB), attributes(5)`
The `mlib_ImageMean()` function computes the mean value of all the pixels in the image. It uses the following equation:

\[
\text{mean}[i] = \frac{1}{w \times h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]
\]

**Parameters**
The function takes the following arguments:

- **mean** Pointer to mean array, where length is the number of channels in the image. `mean[i]` contains the mean of channel `i`.
- **img** Pointer to an image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageMean_Fp(3MLIB)`, `mlib_ImageStdDev(3MLIB)`, `mlib_ImageStdDev_Fp(3MLIB)`, `attributes(5)`
mlib_ImageMean_Fp(3MLIB)

Name  mlib_ImageMean_Fp – image mean

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMean_Fp(mlib_d64 *mean, const mlib_image *img);

Description  The mlib_ImageMean_Fp() function computes the mean value of all the pixels in the image.

It uses the following equation:

\[
\text{mean}[i] = \frac{1}{w \times h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]
\]

Parameters  The function takes the following arguments:

- mean  Pointer to mean array, where length is the number of channels in the image. mean[i] contains the mean of channel i.
- img  Pointer to an image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageMean(3MLIB), mlib_ImageStdDev(3MLIB), mlib_ImageStdDev_Fp(3MLIB), attributes(5)
mlib_ImageMedianFilter3x3(3MLIB)

Name  mlib_ImageMedianFilter3x3 - 3x3 median filter

Synopsis  cc [ flag... ] file... -tllib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMedianFilter3x3(mlib_image *dst,
   const mlib_image *src, mlib_median_mask mmask, mlib_s32 cmask,
   mlib_edge edge);

Description  The mlib_ImageMedianFilter3x3() function performs median filtering on an image. Each
pixel of the destination image is the pixel with rank middle in the filter window.

Parameters  The function takes the following arguments:

dst  Pointer to destination image.

src  Pointer to source image.

mmask  Shape of the mask to be used for median filtering. It can be one of the following:
MLIB_MEDIAN_MASK_RECT
MLIB_MEDIAN_MASK_PLUS
MLIB_MEDIAN_MASK_X
MLIB_MEDIAN_MASK_RECT_SEPARABLE

cmask  Channel mask to indicate the channels to be filtered. Each bit of which represents
a channel in the image. The channels corresponded to 1 bits are those to be
processed.

edge  Type of edge condition. It can be one of the following:
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_DST_COPY_SRC
MLIB_EDGE_SRC_EXTEND

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter3x3_US(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB), mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
Name  mlib ImageMedianFilter3x3_Fp – 3x3 median filter

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>
          mlib_status mlib ImageMedianFilter3x3_Fp(mlib_image *dst,
                          const mlib_image *src, mlib_median_mask mmask, mlib_s32 cmask,
                          mlib_edge edge);

Description  The mlib ImageMedianFilter3x3_Fp() function performs floating-point median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

Parameters  The function takes the following arguments:

  dst  Pointer to destination image.

  src  Pointer to source image.

  mmask  Shape of the mask to be used for median filtering. It can be one of the following:

         MLIB_MEDIAN_MASK_RECT
         MLIB_MEDIAN_MASK_PLUS
         MLIB_MEDIAN_MASK_X
         MLIB_MEDIAN_MASK_RECT_SEPARABLE

  cmask  Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.

  edge  Type of edge condition. It can be one of the following:

         MLIB_EDGE_DST_NO_WRITE
         MLIB_EDGE_DST_FILL_ZERO
         MLIB_EDGE_DST_COPY_SRC
         MLIB_EDGE_SRC_EXTEND

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib ImageMaxFilter3x3(3MLIB), mlib ImageMaxFilter3x3_Fp(3MLIB),
          mlib ImageMaxFilter5x5(3MLIB), mlib ImageMaxFilter5x5_Fp(3MLIB),
          mlib ImageMaxFilter7x7(3MLIB), mlib ImageMaxFilter7x7_Fp(3MLIB),
          mlib ImageMedianFilter3x3(3MLIB), mlib ImageMedianFilter3x3_US(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMedianFilter3x3_US` function performs median filtering on a MLIB_SHORT type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

### Parameters
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **mmask** Shape of the mask to be used for median filtering. It can be one of the following:
  - MLIB_MEDIAN_MASK_RECT
  - MLIB_MEDIAN_MASK_PLUS
  - MLIB_MEDIAN_MASK_X
  - MLIB_MEDIAN_MASK_RECT_SEPARABLE
- **cmask** Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- **edge** Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND
- **bits** The number of unsigned bits for pixel dynamic range. $9 \leq \text{bits} \leq 15$.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
### mlib_ImageMedianFilter5x5(3MLIB)

<table>
<thead>
<tr>
<th>Name</th>
<th>mlib_ImageMedianFilter5x5 - 5x5 median filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>cc [ flag... ] file... -mlib [ library... ]</td>
</tr>
<tr>
<td></td>
<td>#include &lt;mlib.h&gt;</td>
</tr>
<tr>
<td></td>
<td>mlib_status mlib_ImageMedianFilter5x5(mlib_image *dst,</td>
</tr>
<tr>
<td></td>
<td>const mlib_image *src, mlib_median_mask mmask, mlib_s32 cmask,</td>
</tr>
<tr>
<td></td>
<td>mlib_edge edge);</td>
</tr>
<tr>
<td>Description</td>
<td>The <code>mlib_ImageMedianFilter5x5()</code> function performs median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.</td>
</tr>
<tr>
<td>Parameters</td>
<td>The function takes the following arguments:</td>
</tr>
<tr>
<td>dst</td>
<td>Pointer to destination image.</td>
</tr>
<tr>
<td>src</td>
<td>Pointer to source image.</td>
</tr>
<tr>
<td>mmask</td>
<td>Shape of the mask to be used for median filtering. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>MLIB_MEDIAN_MASK_RECT</td>
</tr>
<tr>
<td></td>
<td>MLIB_MEDIAN_MASK_PLUS</td>
</tr>
<tr>
<td></td>
<td>MLIB_MEDIAN_MASK_X</td>
</tr>
<tr>
<td></td>
<td>MLIB_MEDIAN_MASK_RECT_SEPARABLE</td>
</tr>
<tr>
<td>cmask</td>
<td>Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.</td>
</tr>
<tr>
<td>edge</td>
<td>Type of edge condition. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>MLIB_EDGE_DST_NO_WRITE</td>
</tr>
<tr>
<td></td>
<td>MLIB_EDGE_DST_FILL_ZERO</td>
</tr>
<tr>
<td></td>
<td>MLIB_EDGE_DST_COPY_SRC</td>
</tr>
<tr>
<td></td>
<td>MLIB_EDGE_SRC_EXTEND</td>
</tr>
<tr>
<td>Return Values</td>
<td>The function returns <code>MLIB_SUCCESS</code> if successful. Otherwise it returns <code>MLIB_FAILURE</code>.</td>
</tr>
<tr>
<td>Attributes</td>
<td>See attributes(5) for descriptions of the following attributes:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
- mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
- mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
- mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
- mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_US(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_US(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageMinFilter7x7_US(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageMedianFilter5x5_Fp(3MLIB)

Name  mlib_ImageMedianFilter5x5_Fp – 5x5 median filter

Synopsis  cc [ flag... ] file... -mlib [ library... ]
          #include <mlib.h>

          mlib_status mlib_ImageMedianFilter5x5_Fp(mlib_image *dst,
                          const mlib_image *src, mlib_median_mask mmask, mlib_s32 cmask,
                          mlib_edge edge);

Description  The mlib_ImageMedianFilter5x5_Fp() function performs floating-point median filtering
              on an image. Each pixel of the destination image is the pixel with rank middle in the filter
              window.

Parameters  The function takes the following arguments:

              dst   Pointer to destination image.
              src   Pointer to source image.
              mmask Shape of the mask to be used for median filtering. It can be one of the following:
                     MLIB_MEDIAN_MASK_RECT
                     MLIB_MEDIAN_MASK_PLUS
                     MLIB_MEDIAN_MASK_X
                     MLIB_MEDIAN_MASK_RECT_SEPARABLE
              cmask Channel mask to indicate the channels to be filtered. Each bit of which represents
                     a channel in the image. The channels corresponded to 1 bits are those to be
                     processed.
              edge  Type of edge condition. It can be one of the following:
                     MLIB_EDGE_DST_NO_WRITE
                     MLIB_EDGE_DST_FILL_ZERO
                     MLIB_EDGE_DST_COPY_SRC
                     MLIB_EDGE_SRC_EXTEND

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
           mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
           mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
           mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB)

mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMedianFilter5x5_US()` function performs median filtering on an `MLIB_SHORT` type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

### Parameters
- **`dst`**: Pointer to destination image.
- **`src`**: Pointer to source image.
- **`mmask`**: Shape of the mask to be used for median filtering. It can be one of the following:
  - `MLIB_MEDIAN_MASK_RECT`
  - `MLIB_MEDIAN_MASK_PLUS`
  - `MLIB_MEDIAN_MASK_X`
  - `MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- **`cmask`**: Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- **`edge`**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`
- **`bits`**: The number of unsigned bits for pixel dynamic range. $9 \leq \text{bits} \leq 15$.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:
See Also

mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter7x7(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
### Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageMedianFilter7x7(mlib_image *dst,
    const mlib_image *src, mlib_median_mask mmask, mlib_s32 cmask,
    mlib_edge edge);
```

### Description

The `mlib_ImageMedianFilter7x7()` function performs median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **mmask**: Shape of the mask to be used for median filtering. It can be one of the following:
  - `MLIB_MEDIAN_MASK_RECT`
  - `MLIB_MEDIAN_MASK_PLUS`
  - `MLIB_MEDIAN_MASK_X`
  - `MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- **cmask**: Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageMaxFilter3x3(3MLIB)`
- `mlib_ImageMaxFilter3x3_Fp(3MLIB)`
- `mlib_ImageMaxFilter5x5(3MLIB)`
- `mlib_ImageMaxFilter5x5_Fp(3MLIB)`
- `mlib_ImageMaxFilter7x7(3MLIB)`
- `mlib_ImageMaxFilter7x7_Fp(3MLIB)`
- `mlib_ImageMedianFilter3x3(3MLIB)`
- `mlib_ImageMedianFilter3x3_Fp(3MLIB)`
- `mlib_ImageMedianFilter3x3_US(3MLIB)`
- `mlib_ImageMedianFilter5x5(3MLIB)`
- `mlib_ImageMedianFilter5x5_Fp(3MLIB)`
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMedianFilter7x7_Fp()` function performs floating-point median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `mmask` Shape of the mask to be used for median filtering. It can be one of the following:
  - `MLIB_MEDIAN_MASK_RECT`
  - `MLIB_MEDIAN_MASK_PLUS`
  - `MLIB_MEDIAN_MASK_X`
  - `MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- `cmask` Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`, `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`, `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB),
mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB),
mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB),
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMedianFilter7x7_US()` function performs median filtering on an 
MLIB_SHORT type of image that contains unsigned data. Each pixel of the destination image is 
the pixel with rank middle in the filter window.

**Parameters**
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

- **mmask**  
  Shape of the mask to be used for median filtering. It can be one of the following:

  - `MLIB_MEDIAN_MASK_RECT`
  - `MLIB_MEDIAN_MASK_PLUS`
  - `MLIB_MEDIAN_MASK_X`
  - `MLIB_MEDIAN_MASK_RECT_SEPARABLE`

- **cmask**  
  Channel mask to indicate the channels to be filtered. Each bit of which represents 
a channel in the image. The channels corresponded to 1 bits are those to be processed.

- **edge**  
  Type of edge condition. It can be one of the following:

  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

- **bits**  
  The number of unsigned bits for pixel dynamic range. $9 \leq \text{bits} \leq 15$.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
mlib_ImageMaxFilter3x3 (3MLIB), mlib_ImageMaxFilter5x5 (3MLIB), mlib_ImageMaxFilter7x7 (3MLIB),
mlib_ImageMaxFilter3x3_Fp (3MLIB), mlib_ImageMaxFilter5x5_Fp (3MLIB), mlib_ImageMaxFilter7x7_Fp (3MLIB),
mlib_ImageMedianFilter3x3 (3MLIB), mlib_ImageMedianFilter3x3_Fp (3MLIB), mlib_ImageMedianFilter5x5 (3MLIB),
mlib_ImageMedianFilter5x5_Fp (3MLIB), mlib_ImageMedianFilter5x5_US (3MLIB), mlib_ImageMedianFilter7x7 (3MLIB),
mlib_ImageMedianFilter7x7_Fp (3MLIB), mlib_ImageMedianFilterMxN (3MLIB), mlib_ImageMedianFilterMxN_Fp (3MLIB),
mlib_ImageMedianFilterMxN_US (3MLIB), mlib_ImageMinFilter3x3 (3MLIB), mlib_ImageMinFilter5x5 (3MLIB),
mlib_ImageMinFilter7x7 (3MLIB), mlib_ImageMinFilterMxN (3MLIB), mlib_ImageMinFilterMxN_Fp (3MLIB),
mlib_ImageRankFilter3x3 (3MLIB), mlib_ImageRankFilter3x3_Fp (3MLIB), mlib_ImageRankFilter3x3_US (3MLIB),
mlib_ImageRankFilter5x5 (3MLIB), mlib_ImageRankFilter5x5_Fp (3MLIB), mlib_ImageRankFilter7x7 (3MLIB),
mlib_ImageRankFilter7x7_Fp (3MLIB), mlib_ImageRankFilter7x7_US (3MLIB), mlib_ImageRankFilterMxN (3MLIB),
mlib_ImageRankFilterMxN_Fp (3MLIB), mlib_ImageRankFilterMxN_US (3MLIB), attributes(5)}
# mlib_ImageMedianFilterMxN

The `mlib_ImageMedianFilterMxN()` function performs MxN median filtering on an image. Each pixel of the destination image is the pixel with rank middle in the filter window.

### Parameters

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **m**: Width of the filter window. `m` must be odd number greater than 1.
- **n**: Height of the filter window. `n` must be odd number greater than 1.
- **mmask**: Shape of the mask to be used for median filtering. It can be one of the following:
  - MLIB_MEDIAN_MASK_RECT
  - MLIB_MEDIAN_MASK_PLUS
  - MLIB_MEDIAN_MASK_X
  - MLIB_MEDIAN_MASK_RECT_SEPARABLE
- **cmask**: Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also

mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter5x5(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMedianFilterMxN_Fp()` function performs MxN median filtering on a floating-point image. Each pixel of the destination image is the pixel with rank middle in the filter window.

### Parameters
The function takes the following arguments:
- `dst`: Pointer to destination image.
- `src`: Pointer to source image.
- `m`: Width of the filter window. `m` must be odd number greater than 1.
- `n`: Height of the filter window. `n` must be odd number greater than 1.
- `mmask`: Shape of the mask to be used for median filtering. It can be one of the following:
  - `MLIB_MEDIAN_MASK_RECT`
  - `MLIB_MEDIAN_MASK_PLUS`
  - `MLIB_MEDIAN_MASK_X`
  - `MLIB_MEDIAN_MASK_RECT_SEPARABLE`
- `cmask`: Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.
- `edge`: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilterMxN(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),
mlib_ImageMinFilterMxN(3MLIB), mlib_ImageMinFilterMxN_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
# Description
The `mlib_ImageMedianFilterMxN_US()` function performs MxN median filtering on an MLIB_SHORT type of image that contains unsigned data. Each pixel of the destination image is the pixel with rank middle in the filter window.

## Parameters
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

- **m**  
  Width of the filter window. m must be odd number greater than 1.

- **n**  
  Height of the filter window. n must be odd number greater than 1.

- **mmask**  
  Shape of the mask to be used for median filtering. It can be one of the following:
  
  - `MLIB_MEDIAN_MASK_RECT`
  - `MLIB_MEDIAN_MASK_RECT_SEPARABLE`
  - `MLIB_MEDIAN_MASK_PLUS`
  - `MLIB_MEDIAN_MASK_X`

- **cmask**  
  Channel mask to indicate the channels to be filtered. Each bit of which represents a channel in the image. The channels corresponded to 1 bits are those to be processed.

- **edge**  
  Type of edge condition. It can be one of the following:
  
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

- **bits**  
  The number of unsigned bits for pixel dynamic range. $9 \leq \text{bits} \leq 15$.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>ATTRIBUTE TYPE</td>
<td>ATTRIBUTE VALUE</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: `mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB), mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB), mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB), mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)`
The mlib_ImageMin() function accepts input from the two source images and writes the minimum to the destination image on a pixel-by-pixel basis. It uses the following equation:

\[
dst[x][y][i] = \text{MIN\{ src1}[x][y][i], src2[x][y][i] }\]

The function takes the following arguments:

- \textit{dst} Pointer to destination image.
- \textit{src1} Pointer to first source image.
- \textit{src2} Pointer to second source image.

The function returns \text{MLIB\_SUCCESS} if successful. Otherwise it returns \text{MLIB\_FAILURE}.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib\_ImageMin\_Fp(3MLIB), mlib\_ImageMin\_Fp\_Inp(3MLIB), mlib\_ImageMin\_Inp(3MLIB), attributes(5)
The `mlib_ImageMinFilter3x3()` function replaces the center pixel in a neighborhood with the minimum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
dst[x][y][0] = \text{MIN} \{ \text{src}[p][q][0], \quad x-1 \leq p \leq x+1; \quad y-1 \leq q \leq y+1 \}
\]

where \( x = 1, \ldots, w - 2; \quad y = 1, \ldots, h - 2. \)

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`, `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`, `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`
- `mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`, `mlib_ImageMedianFilter3x3_US(3MLIB)`
- `mlib_ImageMedianFilter5x5(3MLIB)`, `mlib_ImageMedianFilter5x5_Fp(3MLIB)`, `mlib_ImageMedianFilter5x5_US(3MLIB)`
- `mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`, `mlib_ImageMedianFilter7x7_US(3MLIB)`
- `mlib_ImageMedianFilterMxN(3MLIB)`, `mlib_ImageMedianFilterMxN_Fp(3MLIB)`
- `mlib_ImageMinFilter3x3(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`
- `mlib_ImageMinFilter5x5(3MLIB)`, `mlib_ImageMinFilter5x5_Fp(3MLIB)`
- `mlib_ImageMinFilter7x7(3MLIB)`, `mlib_ImageMinFilter7x7_Fp(3MLIB)`
- `mlib_ImageRankFilter3x3_Fp(3MLIB)`, `mlib_ImageRankFilter3x3_US(3MLIB)`
- `mlib_ImageRankFilter5x5(3MLIB)`, `mlib_ImageRankFilter5x5_Fp(3MLIB)`
mlib_ImageMinFilter3x3(3MLIB)

mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageMinFilter3x3_Fp (3MLIB)

**Name**  
mlib_ImageMinFilter3x3_Fp – 3x3 Min Filter, floating point

**Synopsis**  
cc [ flag...] file... -lmlib [ library... ]  
#include <mlib.h>  

```c
mlib_status mlib_ImageMinFilter3x3_Fp(mlib_image *dst,
const mlib_image *src);
```

**Description**  
The `mlib_ImageMinFilter3x3_Fp()` function replaces the center pixel in a neighborhood with the floating-point minimum value in that neighborhood for each 3x3 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
dst[x][y][0] = \text{MIN} \{ \text{src}[p][q][0],
\quad x-1 \leq p \leq x+1; \quad y-1 \leq q \leq y+1 \}
\]

where \( x = 1, \ldots, w - 2; \quad y = 1, \ldots, h - 2 \).

**Parameters**  
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),  
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),  
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),  
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),  
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),  
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),  
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),  
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB),  
mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),  
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter5x5(3MLIB),  
mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinMaxFilter7x7(3MLIB),  
mlib_ImageMinMaxFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageMinFilter5x5 - 5x5 Min Filter

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMinFilter5x5(mlib_image *dst, const mlib_image *src);

Description

The mlib_ImageMinFilter5x5() function replaces the center pixel in a neighborhood with the minimum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

dst[x][y][0] = MIN{ src[p][q][0],
    x-2 ≤ p ≤ x+2; y-2 ≤ q ≤ y+2 }

where x = 2, ..., w - 3; y = 2, ..., h - 3.

Parameters

The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB),
mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB),
mlib_ImageRankFilter5x5_Fp(3MLIB),

Multimedia Library Functions - Part 3 507
mlib_ImageMinFilter5x5(3MLIB)

mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
**Name**  
mlib_ImageMinFilter5x5_Fp – 5x5 Min Filter, floating point

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_ImageMinFilter5x5_Fp(mlib_image *dst,  
const mlib_image *src);

**Description**  
The `mlib_ImageMinFilter5x5_Fp()` function replaces the center pixel in a neighborhood with the floating-point minimum value in that neighborhood for each 5x5 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
\text{dst}[x][y][0] = \text{MIN} \{ \text{src}[p][q][0], \  
x-2 \leq p \leq x+2; \ y-2 \leq q \leq y+2 \} \\
\]

where \( x = 2, \ldots, w - 3; \ y = 2, \ldots, h - 3. \)

**Parameters**  
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),  
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),  
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),  
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),  
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),  
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),  
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),  
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB),  
mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),  
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),  
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter7x7(3MLIB),  
mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB),
mlib_ImageMinFilter5x5_Fp(3MLIB)

mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageminFilter7x7()` function replaces the center pixel in a neighborhood with the minimum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

\[
dst[x][y][0] = \text{MIN}\{ \text{src}[p][q][0], \ x-3 \leq p \leq x+3; y-3 \leq q \leq y+3 \}
\]

where \( x = 3, \ldots, w - 4; y = 3, \ldots, h - 4 \).

### Parameters
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageminFilter3x3(3MLIB)`, `mlib_ImageminFilter3x3_Fp(3MLIB)`,
- `mlib_ImageminFilter5x5(3MLIB)`, `mlib_ImageminFilter5x5_Fp(3MLIB)`,
- `mlib_ImageminFilter7x7(3MLIB)`, `mlib_ImageminFilter7x7_Fp(3MLIB)`,
- `mlib_ImagemedianFilter3x3(3MLIB)`, `mlib_ImagemedianFilter3x3_Fp(3MLIB)`,
- `mlib_ImagemedianFilter3x3_US(3MLIB)`, `mlib_ImagemedianFilter5x5(3MLIB)`,
- `mlib_ImagemedianFilter5x5_us(3MLIB)`, `mlib_ImagemedianFilter5x5_Fp(3MLIB)`,
- `mlib_ImagemedianFilter5x5_us(3MLIB)`, `mlib_ImagemedianFilterMxN(3MLIB)`,
- `mlib_ImagemedianFilterMxNFp(3MLIB)`, `mlib_ImagemedianFilterMxN_us(3MLIB)`,
- `mlib_ImageminFilter3x3(3MLIB)`, `mlib_ImageminFilter3x3_Fp(3MLIB)`,
- `mlib_ImageminFilter5x5(3MLIB)`, `mlib_ImageminFilter5x5_Fp(3MLIB)`,
- `mlib_ImageminFilter7x7_Fp(3MLIB)`, `mlib_ImageminFilter3x3_Fp(3MLIB)`,
- `mlib_ImageminFilter5x5(3MLIB)`, `mlib_ImageminFilter5x5_Fp(3MLIB)`,
- `mlib_ImageminFilter7x7_Fp(3MLIB)`, `mlib_ImageminFilter3x3_Fp(3MLIB)`,
- `mlib_ImageminFilter5x5(3MLIB)`, `mlib_ImageminFilter5x5_Fp(3MLIB)`,
- `mlib_ImageminFilter7x7_Fp(3MLIB)`, `mlib_ImageminFilter3x3_Fp(3MLIB)`,
- `mlib_ImageminFilter5x5(3MLIB)`, `mlib_ImageminFilter5x5_Fp(3MLIB)`. 
mlib_ImageMinFilter7x7(3MLIB)

mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
**Synopsis**

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMinFilter7x7_Fp(mlib_image *dst,
                                   const mlib_image *src);
```

**Description**

The `mlib_ImageMinFilter7x7_Fp()` function replaces the center pixel in a neighborhood with the floating-point minimum value in that neighborhood for each 7x7 neighborhood in the image.

The source and destination images must be single-channel images.

It uses the following equation:

$$
dst[x][y][0] = \text{MIN} \{ \text{src}[p][q][0], \quad x-3 \leq p \leq x+3; \quad y-3 \leq q \leq y+3 \}$$

where $$x = 3, \ldots, w - 4; \quad y = 3, \ldots, h - 4$$.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3_US(3MLIB)`, `mlib_ImageMedianFilter5x5(3MLIB)`,
- `mlib_ImageMedianFilter5x5_Fp(3MLIB)`, `mlib_ImageMedianFilter5x5_US(3MLIB)`,
- `mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter7x7_US(3MLIB)`, `mlib_ImageMedianFilterMxN(3MLIB)`,
- `mlib_ImageMedianFilterMxN_Fp(3MLIB)`, `mlib_ImageMedianFilterMxN_US(3MLIB)`,
- `mlib_ImageMinFilter3x3(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMinFilter5x5(3MLIB)`, `mlib_ImageMinFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMinFilter7x7(3MLIB)`, `mlib_ImageRankFilter3x3(3MLIB)`,
- `mlib_Imfilter3x3(3MLIB)`
mlib_ImageMinFilter7x7_Fp(3MLIB)

mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageMin_Fp()` function accepts input from the two floating-point source images and writes the minimum to the destination image on a pixel-by-pixel basis. It uses the following equation:

\[
\text{dst}[x][y][i] = \text{MIN} \{ \text{src1}[x][y][i], \text{src2}[x][y][i] \}
\]

**Parameters**

The function takes the following arguments:

- **`dst`** Pointer to destination image.
- **`src1`** Pointer to first source image.
- **`src2`** Pointer to second source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageMin(3MLIB), mlib_ImageMin_Fp_Inp(3MLIB), mlib_ImageMin_Inp(3MLIB), attributes(5)`
Name  
mlib_ImageMin_Fp_Inp – two-image minimum

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_IMAGE_MIN_Fp_Inp(mlib_image *src1dst,
const mlib_image *src2);

Description  
The *mlib_IMAGE_MIN_Fp_Inp() function accepts input from the two source images and writes
the minimum in place on a pixel-by-pixel basis.

It uses the following equation:
src1dst[x][y][i] = MIN{ src1dst[x][y][i], src2[x][y][i] }

Parameters  
The function takes the following arguments:
src1dst  Pointer to source and destination image.
src2  Pointer to second source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_IMAGE_MIN(3MLIB), mlib_IMAGE_MIN_Fp(3MLIB), mlib_IMAGE_MIN_Inp(3MLIB), attributes(5)
mlib_ImageMinimum() function determines the minimum value for each channel in an image.

It uses the following equation:

\[ \text{min}[i] = \min \{ \text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h \} \]

**Parameters**

- \( \text{min} \): Pointer to minimum vector, where length is the number of channels in the image. \( \text{min}[i] \) contains the minimum of channel \( i \).
- \( \text{img} \): Pointer to a source image.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_ImageMaximum(3MLIB), mlib_ImageMaximum_Fp(3MLIB), mlib_ImageMinimum_Fp(3MLIB), attributes(5)
**Name**
mlib_ImageMinimum_Fp – image minimum

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_ImageMinimum_Fp(mlib_d64 *min, const mlib_image *img);
```

**Description**
The `mlib_ImageMinimum_Fp()` function determines the minimum value for each channel in a floating-point image.

It uses the following equation:

\[
\text{min}[i] = \text{MIN} \{ \text{img}[x][y][i]; \ 0 \leq x < w, \ 0 \leq y < h \}
\]

**Parameters**
The function takes the following arguments:

- `min` Pointer to minimum vector, where length is the number of channels in the image. `min[i]` contains the minimum of channel `i`.
- `img` Pointer to a source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageMaximum(3MLIB)`, `mlib_ImageMaximum_Fp(3MLIB)`
- `mlib_ImageMinimum(3MLIB)`, attributes(5)
The `mlib_ImageMin_Inp()` function accepts input from the two source images and writes the minimum in place on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{src1dst}[x][y][i] = \text{MIN}\{ \text{src1dst}[x][y][i], \text{src2}[x][y][i] \}
\]

**Parameters**
- `src1dst` Pointer to source and destination image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageMin(3MLIB), mlib_ImageMin_Fp(3MLIB), mlib_ImageMin_Fp_Inp(3MLIB), attributes(5)`
# mlib_ImageMoment2

## Synopsis
```
c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMoment2(mlib_d64 *moment, const mlib_image *img);
```

## Description
The `mlib_ImageMoment2()` function computes the second moment of each channel in an image.

It uses the following equation:
\[
\text{moment}[i] = \frac{1}{w \times h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]^2
\]

## Parameters
- **moment**: Pointer to moment array, where length is the number of channels in the image.
- **img**: Pointer to an image.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageMoment2_Fp(3MLIB)`, `attributes(5)`
The `mlib_ImageMoment2_Fp()` function computes the second moment of each channel in a floating-point image.

It uses the following equation:

\[
\text{moment}[i] = \frac{1}{w \cdot h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]^2
\]

The function takes the following arguments:

- `moment` Pointer to moment array, where length is the number of channels in the image.
- `img` Pointer to an image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageMoment2(3MLIB), attributes(5)`
Name  mlib_ImageMulAlpha – alpha channel multiplication

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMulAlpha(const mlib_image *dst, const mlib_image *src,
                               mlib_s32 cmask);

Description The mlib_ImageMulAlpha() function multiplies color channels by the alpha channel on a
pixel by pixel basis.

For the MLIB_BYTE image, it uses the following equation:
\[ \text{dst}[x][y][c] = \text{src}[x][y][c] \times \text{src}[x][y][a] \times 2^{-8} \]

For the MLIB_SHORT image, it uses the following equation:
\[ \text{dst}[x][y][c] = \text{src}[x][y][c] \times \text{src}[x][y][a] \times 2^{-15} \]

For the MLIB_USHORT image, it uses the following equation:
\[ \text{dst}[x][y][c] = \text{src}[x][y][c] \times \text{src}[x][y][a] \times 2^{-16} \]

For the MLIB_INT image, it uses the following equation:
\[ \text{dst}[x][y][c] = \text{src}[x][y][c] \times \text{src}[x][y][a] \times 2^{-31} \]

where \( c \) and \( a \) are the indices for the color channels and the alpha channel, respectively, so \( c \neq a \).

Parameters The function takes the following arguments:

- \( \text{dst} \) Pointer to destination image.
- \( \text{src} \) Pointer to source image.
- \( \text{cmask} \) Channel mask to indicate the alpha channel. Each bit of the mask represents a
  channel in the image. The channel corresponding to the 1 bit of \( \text{cmask} \) is the alpha
  channel.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_ImageMulAlpha_Inp(3MLIB), mlib_ImageMulAlpha_Fp(3MLIB),  
mlib_ImageMulAlpha_Fp_Inp(3MLIB), attributes(5)
mlib_ImageMulAlpha_Fp – alpha channel multiplication

### Synopsis
```c
#include <mlib.h>

mlib_status mlib_ImageMulAlpha_Fp(mlib_image *dst, const mlib_image *src, 
        mlib_s32 cmask);
```

### Description
The `mlib_ImageMulAlpha_Fp()` function multiplies floating-point color channels by the alpha channel on a pixel by pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][c] = \text{src}[x][y][c] \times \text{src}[x][y][a] \]

where \( c \) and \( a \) are the indices for the color channels and the alpha channel, respectively, so \( c \neq a \).

### Parameters
The function takes the following arguments:
- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **cmask** Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageMulAlpha(3MLIB)`
- `mlib_ImageMulAlpha_Inp(3MLIB)`
- `mlib_ImageMulAlpha_Fp_Inp(3MLIB)`, `attributes(5)`
mlib_ImageMulAlpha_Fp_Inp() function multiplies floating-point color channels by the alpha channel on a pixel by pixel basis, in place.

It uses the following equation:

\[ \text{srcdst}[x][y][c] = \text{srcdst}[x][y][c] \times \text{srcdst}[x][y][a] \]

where \( c \) and \( a \) are the indices for the color channels and the alpha channel, respectively, so \( c \neq a \).

### Parameters

- **srcdst** Pointer to source and destination image.
- **cmask** Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of cmask is the alpha channel.

### Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageMulAlpha(3MLIB), mlib_ImageMulAlpha_Inp(3MLIB), mlib_ImageMulAlpha_Fp(3MLIB), attributes(5)
The `mlib_ImageMulAlpha_Inp()` function multiplies color channels by the alpha channel on a pixel by pixel basis, in place.

For the `MLIB_BYTE` image, it uses the following equation:

```c
srcdst[x][y][c] = srcdst[x][y][c] * srcdst[x][y][a] * 2**(-8)
```

For the `MLIB_SHORT` image, it uses the following equation:

```c
srcdst[x][y][c] = srcdst[x][y][c] * srcdst[x][y][a] * 2**(-15)
```

For the `MLIB_USHORT` image, it uses the following equation:

```c
srcdst[x][y][c] = srcdst[x][y][c] * srcdst[x][y][a] * 2**(-16)
```

For the `MLIB_INT` image, it uses the following equation:

```c
srcdst[x][y][c] = srcdst[x][y][c] * srcdst[x][y][a] * 2**(-31)
```

where \( c \) and \( a \) are the indices for the color channels and the alpha channel, respectively, so \( c \neq a \).

**Parameters**

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `cmask` Channel mask to indicate the alpha channel. Each bit of the mask represents a channel in the image. The channel corresponding to the 1 bit of `cmask` is the alpha channel.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_ImageMulAlpha(3MLIB), mlib_ImageMulAlpha_Fp(3MLIB), mlib_ImageMulAlpha_Fp_Inp(3MLIB), attributes(5)`
mlib_ImageMul_Fp(3MLIB)

**Name**
mlib_ImageMul_Fp – computes the multiplication of two images on a pixel-by-pixel basis

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMul_Fp(mlib_image *dst, const mlib_image *src1, const mlib_image *src2);
```

**Description**
The `mlib_ImageMul_Fp()` function computes the multiplication of two floating-point images on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = \text{src1}[x][y][i] \times \text{src2}[x][y][i]
\]

**Parameters**
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageMul_Fp_Inp(3MLIB), attributes(5)`
mlib_ImageMul_Fp_Inp (3MLIB)

**Name**
mlib_ImageMul_Fp_Inp – computes the multiplication of two images on a pixel-by-pixel basis

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageMul_Fp_Inp(mlib_image *src1dst,
const mlib_image *src2);

**Description**
The `mlib_ImageMul_Fp_Inp()` function computes the multiplication of two floating-point images on a pixel-by-pixel basis, in place.

It uses the following equation:
src1dst[x][y][i] = src1dst[x][y][i] * src2[x][y][i]

**Parameters**
The function takes the following arguments:

- *src1dst* Pointer to source and destination image.
- *src2* Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageMul_Fp(3MLIB), attributes(5)
The `mlib_ImageMulShift()` function multiplies the pixel values of the two source images. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[ \text{dst}[x][y][i] = \text{src1}[x][y][i] \times \text{src2}[x][y][i] \times 2^{-\text{shift}} \]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.
- `shift` Right shifting factor. 0 ≤ shift ≤ 31.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageMulShift_Inp(3MLIB), attributes(5)`
The `mlib_ImageMulShift_Inp()` function multiplies the pixel values of the two source images in place. It scales the result by a right shift and writes the result to the destination image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] \times \text{src2}[x][y][i] \times 2^{(-\text{shift})}
\]

### Parameters

- **src1dst**: Pointer to source and destination image.
- **src2**: Pointer to second source image.
- **shift**: Right shifting factor. \(0 \leq \text{shift} \leq 31\).

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageMulShift(3MLIB), attributes(5)`
The `mlib_ImageNormCrossCorrel()` function computes the normalized cross-correlation coefficients between a pair of images, on a per-channel basis.

It uses the following equations:

\[
\text{correl}[i] = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (d1[x][y][i] \cdot d2[x][y][i])}{s1[i] \cdot s2[i]}
\]

Where \(d1[x][y][i] = \text{img1}[x][y][i] - m1[i]\) and \(d2[x][y][i] = \text{img2}[x][y][i] - m2[i]\).

\[
m1[i] = \frac{1}{w \cdot h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img1}[x][y][i]
\]

\[
m2[i] = \frac{1}{w \cdot h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img2}[x][y][i]
\]

\[
s1[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] - m1[i])^2}
\]

\[
s2[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img2}[x][y][i] - m2[i])^2}
\]

where \(w\) and \(h\) are the width and height of the images, respectively; \(m1\) and \(m2\) are the mean arrays of the first and second images, respectively; \(s1\) and \(s2\) are the un-normalized standard deviation arrays of the first and second images, respectively.

In usual cases, the normalized cross-correlation coefficient is in the range of \([-1.0, 1.0]\). In the case of \((s1[i] == 0)\) or \((s2[i] == 0)\), where a constant image channel is involved, the normalized cross-correlation coefficient is defined as follows:

\[
\#define \text{signof}(x) ((x > 0) \ ? \ 1 : ((x < 0) \ ? \ -1 : \ 0))
\]
if ((s1[i] == 0.) || (s2[i] == 0.)) {
    if ((s1[i] == 0.) && (s2[i] == 0.)) {
        if (signof(m1[i]) == signof(m2[i])) {
            correl[i] = 1.0;
        } else {
            correl[i] = -1.0;
        }
    } else {
        correl[i] = -1.0;
    }
} else {
    correl[i] = -1.0;
}

The two images must have the same type, the same size, and the same number of channels. They can have 1, 2, 3 or 4 channels. They can be of type MLIB_BYTE, MLIB_SHORT, MLIB_USHORT or MLIB_INT.

If (mean2 == NULL) or (sdev2 == NULL), then m2 and s2 are calculated in this function according to the formulas shown above. Otherwise, they are calculated as follows:

\[
m2[i] = \text{mean2}[i];
\]
\[
s2[i] = \text{sdev2}[i] \times \sqrt{w\times h};
\]

where mean2 and sdev2 can be the output of mlib_ImageMean() and mlib_ImageStdDev(), respectively.

**Parameters**
The function takes the following arguments:

- **correl** Pointer to normalized cross correlation array on a channel basis. The array must be the size of channels in the images. correl[i] contains the cross-correlation of channel i.
- **img1** Pointer to first image.
- **img2** Pointer to second image.
- **mean2** Pointer to the mean array of the second image.
- **sdev2** Pointer to the standard deviation array of the second image.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

- `mlib_ImageAutoCorrel(3MLIB)`
- `mlib_ImageAutoCorrel_Fp(3MLIB)`
- `mlib_ImageCrossCorrel(3MLIB)`
- `mlib_ImageCrossCorrel_Fp(3MLIB)`
- `mlib_ImageNormCrossCorrel_Fp(3MLIB)`
- `attributes(5)`
The `mlib_ImageNormCrossCorrel_Fp()` function computes the normalized cross-correlation coefficients between a pair of floating-point images, on a per-channel basis.

It uses the following equations:

\[
\text{correl}[i] = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (d1[x][y][i] \times d2[x][y][i])}{s1[i] \times s2[i]}
\]

\[
d1[x][y][i] = \text{img1}[x][y][i] - m1[i]
\]

\[
d2[x][y][i] = \text{img2}[x][y][i] - m2[i]
\]

\[
m1[i] = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img1}[x][y][i]}{w \times h}
\]

\[
m2[i] = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img2}[x][y][i]}{w \times h}
\]

\[
s1[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img1}[x][y][i] - m1[i])^2}
\]

\[
s2[i] = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img2}[x][y][i] - m2[i])^2}
\]

where \(w\) and \(h\) are the width and height of the images, respectively; \(m1\) and \(m2\) are the mean arrays of the first and second images, respectively; \(s1\) and \(s2\) are the un-normalized standard deviation arrays of the first and second images, respectively.

In usual cases, the normalized cross-correlation coefficient is in the range of \([-1.0, 1.0]\). In the case of \((s1[i] == 0)\) or \((s2[i] == 0)\), where a constant image channel is involved, the normalized cross-correlation coefficient is defined as follows:

\[
\text{signof}(x) = ((x > 0) ? 1 : ((x < 0) ? -1 : 0))
\]
if ((s1[i] == 0.0) || (s2[i] == 0.0)) {
    if ((s1[i] == 0.0) && (s2[i] == 0.0)) {
        if (signof(m1[i]) == signof(m2[i])) {
            correl[i] = 1.0;
        } else {
            correl[i] = -1.0;
        }
    } else {
        correl[i] = -1.0;
    }
} else {
    correl[i] = -1.0;
}

The two images must have the same type, the same size, and the same number of channels. They can have 1, 2, 3 or 4 channels. They can be of type MLIB_FLOAT or MLIB_DOUBLE.

If (mean2 == NULL) or (sdev2 == NULL), then m2 and s2 are calculated in this function according to the formulas shown above. Otherwise, they are calculated as follows:

\[
m2[i] = \text{mean2}[i];
\]
\[
s2[i] = \text{sdev2}[i] \times \sqrt{\text{w} \times \text{h}};
\]

where mean2 and sdev2 can be the output of mlib_ImageMean() and mlib_ImageStdDev(), respectively.

In some cases, the resulting coefficients of this function could be NaN, Inf, or -Inf.

**Parameters** The function takes the following arguments:

- **correl**: Pointer to normalized cross correlation array on a channel basis. The array must be the size of channels in the images. correl[i] contains the cross-correlation of channel i.
- **img1**: Pointer to first image.
- **img2**: Pointer to second image.
- **mean2**: Pointer to the mean array of the second image.
- **sdev2**: Pointer to the standard deviation array of the second image.

**Return Values** The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageAutoCorrel(3MLIB), mlib_ImageAutoCorrel_Fp(3MLIB),
mlib_ImageCrossCorrel(3MLIB), mlib_ImageCrossCorrel_Fp(3MLIB),
mlib_ImageNormCrossCorrel(3MLIB), attributes(5)
The `mlib_ImageNot()` function computes the logical Not of each pixel in the source image. It uses the following equation:

\[
dst[x][y][i] = \neg src[x][y][i]
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageNot_Inp(3MLIB)`, attributes(5)
The `mlib_ImageNotAnd()` function computes the logical AND of the first source image with the second source image and then takes the logical NOT of that result on a pixel-by-pixel basis.

It uses the following equation:

\[
dst[x][y][i] = \neg(src1[x][y][i] \& (src2[x][y][i]))
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageNotAnd_Inp(3MLIB)`, attributes(5)
The `mlib_ImageNotAnd_Inp()` function computes the logical And of the first source image with the second source images and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
src1dst[x][y][i] = ~ (src1dst[x][y][i] & (src2[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageNotAnd(3MLIB), attributes(5)`
The `mlib_ImageNot_Inp()` function computes the logical Not of each pixel in the source image, in place.

It uses the following equation:

```
srcdst[x][y][i] = ~srcdst[x][y][i]
```

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageNot(3MLIB), attributes(5)`
# mlib_ImageNotOr

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageNotOr(mlib_image *dst, const mlib_image *src1,
                           const mlib_image *src2);
```

**Description**
The `mlib_ImageNotOr()` function computes the logical Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:
```
dst[x][y][i] = ~(src1[x][y][i] | (src2[x][y][i])
```

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageNotOr_Inp(3MLIB)`, attributes(5)
The `mlib_ImageNotOr_Inp()` function computes the logical Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{src1dst}[x][y][i] = \neg(\text{src1dst}[x][y][i] | (\text{src2}[x][y][i]))
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

### Parameters

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageNotOr(3MLIB)`, attributes(5)
The `mlib_ImageNotXor()` function computes the logical exclusive Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis.

It uses the following equation:

\[ dst[x][y][i] = \neg(src1[x][y][i] \land src2[x][y][i]) \]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**  The function takes the following arguments:

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_ImageNotXor_Inp(3MLIB)`, attributes(5)
The `mlib_ImageNotXor_Inp()` function computes the logical exclusive Or of the first and second source images and then takes the logical Not of that result on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{src1dst}[x][y][i] = \neg (\text{src1dst}[x][y][i] \oplus \text{src2}[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageNotXor(3MLIB)`, attributes(5)
The `mlib_ImageOr` function computes the logical Or of the first source image with the second source image on a pixel-by-pixel basis. It uses the following equation:

\[ \text{dst}[x][y][i] = \text{src1}[x][y][i] \lor \text{src2}[x][y][i] \]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageOr_Inp(3MLIB)`, attributes(5)
# mlib_ImageOr_Inp

## Synopsis

```
cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageOr_Inp(mlib_image *src1dst, const mlib_image *src2);
```

## Description

The `mlib_ImageOr_Inp()` function computes the logical Or of the first source image with the second source image on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src1dst[x][y][i] = src1dst[x][y][i] | src2[x][y][i]
```

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

## Parameters

- **src1dst**: Pointer to first source and destination image.
- **src2**: Pointer to second source image.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_ImageOr(3MLIB)`, `attributes(5)`
# mlib_ImageOrNot1_Inp

The `mlib_ImageOrNot1_Inp()` function computes the logical Not of the second source image and then takes the logical Or of that result with the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] \lor (\neg \text{src2}[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

## Parameters

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`mlib_ImageOrNot(3MLIB)`, `mlib_ImageOrNot2_Inp(3MLIB)`, `attributes(5)`. 

---

**Name** mlib_ImageOrNot1_Inp - OrNot, in place

**Synopsis**

```c
cc { flag... } file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_ImageOrNot1_Inp(mlib_image *src1dst,
const mlib_image *src2);
```
The `mlib_ImageOrNot2_Inp()` function computes the logical Not of the second source image and then takes the logical Or of that result with the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src2dst[x][y][i] = src1[x][y][i] | (~src2dst[x][y][i])
```

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

### Parameters

The function takes the following arguments:

- `src2dst` Pointer to second source and destination image.
- `src1` Pointer to first source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageOrNot(3MLIB)`, `mlib_ImageOrNot1_Inp(3MLIB)`, `attributes(5)`
The `mlib_ImageOrNot()` function computes the logical Not of the second source image and then takes the logical Or of that result with the first source image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = \text{src1}[x][y][i] \mid (\neg \text{src2}[x][y][i])
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageOrNot1_Inp(3MLIB)`, `mlib_ImageOrNot2_Inp(3MLIB)`, attributes(5)
The `mlib_ImagePolynomialWarp()` function performs a polynomial-based image warp.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`. The source and destination images may have different sizes.

The `xCoeffs` and `yCoeffs` parameters must contain the same number of coefficients of the form \((n + 1)(n + 2)/2\) for some \(n\), where \(n\) is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

\[ 1, x, y, x^2, xy, y^2, \ldots, x^n, x^{n-1}y, \ldots, xy^{n-1}, y^n \]

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at 0.5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point \(D\) is backward mapped to a point \(S\) in the source image. Then the source pixels with their centers surrounding point \(S\) are selected to do one of the interpolations specified by the `filter` parameter to generate the pixel value for point \(D\).

The mapping is defined by the two bivariate polynomial functions \(X(x, y)\) and \(Y(x, y)\) that map destination \((x, y)\) coordinates to source \(X\) and \(Y\) positions respectively.

The functions \(X(x, y)\) and \(Y(x, y)\) are:

\[
\begin{align*}
\text{preX} &= (x + \text{preShiftX}) \times \text{preScaleX} \\
\text{preY} &= (y + \text{preShiftY}) \times \text{preScaleY} \\
\text{warpedX} &= \sum \left( \sum \{ \text{xCoeffs}_{ij} \times \text{preX}^{i-j} \times \text{preY}^{j} \} \right) \\
&\quad \text{for } i=0 \text{ to } n \\
\text{warpedY} &= \sum \left( \sum \{ \text{yCoeffs}_{ij} \times \text{preX}^{i-j} \times \text{preY}^{j} \} \right) \\
&\quad \text{for } j=0 \text{ to } n
\end{align*}
\]

```c
#include <mlib.h>

mlib_status mlib_ImagePolynomialWarp(mlib_image *dst, const mlib_image *src, const mlib_d64 *xCoeffs, const mlib_d64 *yCoeffs, mlib_s32 n, mlib_d64 preShiftX, mlib_d64 preShiftY, mlib_d64 postShiftX, mlib_d64 postShiftY, mlib_d64 preScaleX, mlib_d64 preScaleY, mlib_d64 postScaleX, mlib_d64 postScaleY, mlib_filter filter, mlib_edge edge);
```
The destination \((x, y)\) coordinates are pre-shifted by \((\text{preShiftX}, \text{preShiftY})\) and pre-scaled by the factors \(\text{preScaleX}\) and \(\text{preScaleY}\) prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by \(\text{postScaleX}\) and \(\text{postScaleY}\), and then shifted by \((-\text{postShiftX}, -\text{postShiftY})\) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **xCoeffs**: Destination to source transform coefficients for the X coordinate.
- **yCoeffs**: Destination to source transform coefficients for the Y coordinate.
- **n**: Degree power of the polynomial.
- **preShiftX**: Displacement to apply to destination X positions.
- **preShiftY**: Displacement to apply to destination Y positions.
- **postShiftX**: Displacement to apply to source X positions.
- **postShiftY**: Displacement to apply to source Y positions.
- **preScaleX**: Scale factor to apply to destination X positions.
- **preScaleY**: Scale factor to apply to destination Y positions.
- **postScaleX**: Scale factor to apply to source X positions.
- **postScaleY**: Scale factor to apply to source Y positions.
- **filter**: Type of resampling filter. It can be one of the following:
  - MLIB_NEAREST
  - MLIB_BILINEAR
  - MLIB_BICUBIC
  - MLIB_BICUBIC2
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_SRC_PADDED
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_ImagePolynomialWarp(3MLIB), mlib_ImagePolynomialWarpTable(3MLIB), mlib_ImagePolynomialWarpTable_Fp(3MLIB), attributes(5)`
The `mlib_ImagePolynomialWarp_Fp()` function performs a polynomial-based image warp on a floating-point image.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The source and destination images may have different sizes.

The `xCoeffs` and `yCoeffs` parameters must contain the same number of coefficients of the form \((n + 1)(n + 2)/2\) for some \(n\), where \(n\) is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

1, \(x\), \(x^2\), \(x^y\), \(y^2\), ..., \(x^n\), \(x^{(n-1)}y\), \(x^y^{(n-1)}\), \(y^n\)

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at .5 coordinate points. In other words, the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

For each pixel in the destination image, its center point \(D\) is backward mapped to a point \(S\) in the source image. Then the source pixels with their centers surrounding point \(S\) are selected to do one of the interpolations specified by the `filter` parameter to generate the pixel value for point \(D\).

The mapping is defined by the two bivariate polynomial functions \(X(x, y)\) and \(Y(x, y)\) that map destination \((x, y)\) coordinates to source \(X\) and \(Y\) positions respectively.

The functions \(X(x, y)\) and \(Y(x, y)\) are:

\[
\text{preX} = (x + \text{preShiftX}) \times \text{preScaleX}
\]

\[
\text{preY} = (y + \text{preShiftY}) \times \text{preScaleY}
\]

\[
\text{warpedX} = \sum_{i=0}^{n} \sum_{j=0}^{i} \{\text{xCoeffs}_{ij} \times \text{preX}^{(i-j)} \times \text{preY}^{(j)}\}
\]
\[
\text{warpedY} = \sum_{i=0}^{n} \sum_{j=0}^{i} \{yCoeffs_{ij} \times \text{preX}^{(i-j)} \times \text{preY}^{j}\}
\]

\[
X(x, y) = \text{warpedX} \times \text{postScaleX} - \text{postShiftX}
\]

\[
Y(x, y) = \text{warpedY} \times \text{postScaleY} - \text{postShiftY}
\]

The destination \((x, y)\) coordinates are pre-shifted by \((\text{preShiftX}, \text{preShiftY})\) and pre-scaled by the factors \(\text{preScaleX}\) and \(\text{preScaleY}\) prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by \(\text{postScaleX}\) and \(\text{postScaleY}\), and then shifted by \((-\text{postShiftX}, -\text{postShiftY})\) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters**

The function takes the following arguments:

- *dst* Pointer to destination image.
- *src* Pointer to source image.
- *xCoeffs* Destination to source transform coefficients for the X coordinate.
- *yCoeffs* Destination to source transform coefficients for the Y coordinate.
- *n* Degree power of the polynomial.
- *preShiftX* Displacement to apply to destination X positions.
- *preShiftY* Displacement to apply to destination Y positions.
- *postShiftX* Displacement to apply to source X positions.
- *postShiftY* Displacement to apply to source Y positions.
- *preScaleX* Scale factor to apply to destination X positions.
- *preScaleY* Scale factor to apply to destination Y positions.
- *postScaleX* Scale factor to apply to source X positions.
- *postScaleY* Scale factor to apply to source Y positions.
- *filter* Type of resampling filter. It can be one of the following:
  - MLIB_NEAREST
  - MLIB_BILINEAR
  - MLIB_BICUBIC
  - MLIB_BICUBIC2
- *edge* Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_SRC_PADDED
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImagePolynomialWarp(3MLIB), mlib_ImagePolynomialWarpTable(3MLIB), mlib_ImagePolynomialWarpTable_Fp(3MLIB), attributes(5)
mlib_ImagePolynomialWarpTable – polynomial-based image warp with table-driven interpolation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_ImagePolynomialWarpTable(mlib_image *dst,
const mlib_image *src, const mlib_d64 *xCoeffs,
const mlib_d64 *yCoeffs, mlib_s32 n, mlib_d64 preShiftX,
mlib_d64 preShiftY, mlib_d64 postShiftX, mlib_d64 postShiftY,
mlib_d64 preScaleX, mlib_d64 preScaleY, mlib_d64 postScaleX,
mlib_d64 postScaleY, const void *interp_table, mlib_edge edge);

Description

The mlib_ImagePolynomialWarpTable() function performs a polynomial-based image warp with table-driven interpolation.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT. The source and destination images may have different sizes.

The xCoeffs and yCoeffs parameters must contain the same number of coefficients of the form \((n + 1)(n + 2)/2\) for some \(n\), where \(n\) is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

- \(1, x, y, x^2, x*y, y^2, \ldots\),
- \(x^n, x^{(n-1)}y, \ldots, x*y^{(n-1)}, y^n\)

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at .5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point \(D\) is backward mapped to a point \(S\) in the source image. Then the source pixels with their centers surrounding point \(S\) are selected to do the interpolation specified by interp_table to generate the pixel value for point \(D\).

The mapping is defined by the two bivariate polynomial functions \(X(x, y)\) and \(Y(x, y)\) that map destination \((x, y)\) coordinates to source \(X\) and \(Y\) positions respectively.

The functions \(X(x, y)\) and \(Y(x, y)\) are:

- \(preX = (x + preShiftX)\) \* preScaleX
- \(preY = (y + preShiftY)\) \* preScaleY

\[
warpedX = \sum_{i=0}^{n} \sum_{j=0}^{i} xCoeffs_{ij} \cdot preX^{(i-j)} \cdot preY^{(j)}
\]
\[
\text{warpedY} = \sum_{i=0}^{n} \sum_{j=0}^{i} \{ y\coeffs_{ij} \ast \text{preX}^{(i-j)} \ast \text{preY}^{j} \}
\]

\[X(x, y) = \text{warpedX} \ast \text{postScaleX} - \text{postShiftX}\]

\[Y(x, y) = \text{warpedY} \ast \text{postScaleY} - \text{postShiftY}\]

The destination \((x, y)\) coordinates are pre-shifted by \((\text{preShiftX}, \text{preShiftY})\) and pre-scaled by the factors \(\text{preScaleX}\) and \(\text{preScaleY}\) prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by \(\text{postScaleX}\) and \(\text{postScaleY}\), and then shifted by \((-\text{postShiftX}, -\text{postShiftY})\) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters**

The function takes the following arguments:

- \(\text{dst}\) Pointer to destination image.
- \(\text{src}\) Pointer to source image.
- \(x\coeffs\) Destination to source transform coefficients for the X coordinate.
- \(y\coeffs\) Destination to source transform coefficients for the Y coordinate.
- \(n\) Degree power of the polynomial.
- \(\text{preShiftX}\) Displacement to apply to destination X positions.
- \(\text{preShiftY}\) Displacement to apply to destination Y positions.
- \(\text{postShiftX}\) Displacement to apply to source X positions.
- \(\text{postShiftY}\) Displacement to apply to source Y positions.
- \(\text{preScaleX}\) Scale factor to apply to destination X positions.
- \(\text{preScaleY}\) Scale factor to apply to destination Y positions.
- \(\text{postScaleX}\) Scale factor to apply to source X positions.
- \(\text{postScaleY}\) Scale factor to apply to source Y positions.
- \(\text{interp\_table}\) Pointer to an interpolation table. The table is created by the \(\text{mlib\_ImageInterpTableCreate()}\) function.
- \(\text{edge}\) Type of edge condition. It can be one of the following:
  - \(\text{MLIB\_EDGE\_DST\_NO\_WRITE}\)
  - \(\text{MLIB\_EDGE\_SRC\_PADDED}\)

**Return Values**

The function returns \(\text{MLIB\_SUCCESS}\) if successful. Otherwise it returns \(\text{MLIB\_FAILURE}\).
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageInterpTableCreate(3MLIB), mlib_ImageInterpTableDelete(3MLIB),
mlib_ImagePolynomialWarpTable_Fp(3MLIB), mlib_ImagePolynomialWarp(3MLIB),
mlib_ImagePolynomialWarp_Fp(3MLIB), attributes(5)
The `mlib_ImagePolynomialWarpTable_Fp()` function performs a polynomial-based image warp on a floating-point image with table-driven interpolation.

The images must have the same type, and the same number of channels. The images can have 1, 2, 3, or 4 channels. The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`. The source and destination images may have different sizes.

The `xCoeffs` and `yCoeffs` parameters must contain the same number of coefficients of the form \((n+1)(n+2)/2\) for some \(n\), where \(n\) is the degree power of the polynomial. The coefficients, in order, are associated with the terms:

\[
1, x, y, x^2, xy, y^2, \ldots, \\
x^n, x^{n-1}y, \ldots, xy^{n-1}, y^n
\]

and coefficients of value 0 cannot be omitted.

The image pixels are assumed to be centered at 0.5 coordinate points. In other words, the upper-left corner pixel of an image is located at (0.5, 0.5).

For each pixel in the destination image, its center point \(D\) is backward mapped to a point \(S\) in the source image. Then the source pixels with their centers surrounding point \(S\) are selected to do the interpolation specified by `interp_table` to generate the pixel value for point \(D\).

The mapping is defined by the two bivariate polynomial functions \(X(x, y)\) and \(Y(x, y)\) that map destination \((x, y)\) coordinates to source \(X\) and \(Y\) positions respectively.

The functions \(X(x, y)\) and \(Y(x, y)\) are:

\[
preX = (x + preShiftX)*preScaleX \\
preY = (y + preShiftY)*preScaleY \\
warpedX = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \{xCoeffs_{ij} * preX^{(i-j)} * preY^{j}\}
\]
\[
\text{warped}Y = \sum_{i=0}^{n} \sum_{j=0}^{i} \{y\text{Coeffs}_{ij} \times x\text{pre}^{i-j} \times y\text{pre}^{j}\}
\]

\[
X(x, y) = \text{warped}X \times \text{postScale}X - \text{postShift}X
\]

\[
Y(x, y) = \text{warped}Y \times \text{postScale}Y - \text{postShift}Y
\]

The destination \((x, y)\) coordinates are pre-shifted by \((\text{preShift}X, \text{preShift}Y)\) and pre-scaled by the factors \(\text{preScale}X\) and \(\text{preScale}Y\) prior to the evaluation of the polynomial. The results of the polynomial evaluations are scaled by \(\text{postScale}X\) and \(\text{postScale}Y\), and then shifted by \((-\text{postShift}X, -\text{postShift}Y)\) to produce the source pixel coordinates. This process allows for better precision of the results and supports tiled images.

**Parameters**

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **xCoeffs** Destination to source transform coefficients for the X coordinate.
- **yCoeffs** Destination to source transform coefficients for the Y coordinate.
- **n** Degree power of the polynomial.
- **preShiftX** Displacement to apply to destination X positions.
- **preShiftY** Displacement to apply to destination Y positions.
- **postShiftX** Displacement to apply to source X positions.
- **postShiftY** Displacement to apply to source Y positions.
- **preScaleX** Scale factor to apply to destination X positions.
- **preScaleY** Scale factor to apply to destination Y positions.
- **postScaleX** Scale factor to apply to source X positions.
- **postScaleY** Scale factor to apply to source Y positions.
- **interp_table** Pointer to an interpolation table. The table is created by the `mlib_ImageInterpTableCreate()` function.
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_SRC_PADDED`

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageInterpTableCreate(3MLIB), mlib_ImageInterpTableDelete(3MLIB), mlib_ImagePolynomialWarpTable(3MLIB), mlib_ImagePolynomialWarp(3MLIB), mlib_ImagePolynomialWarp_Fp(3MLIB), attributes(5)
The `mlib_ImageRankFilter3x3()` function performs 3x3 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window. The source and destination images must be single-channel images.

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **rank**: The rank of the destination pixel. The pixel with minimum value is designated rank 0.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMedianFilter5x5(3MLIB)`, `mlib_ImageMedianFilter5x5_Fp(3MLIB)`, `mlib_ImageMedianFilter5x5_US(3MLIB)`,
- `mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`, `mlib_ImageMedianFilter7x7_US(3MLIB)`,
- `mlib_ImageMedianFilterMxN(3MLIB)`, `mlib_ImageMedianFilterMxN_Fp(3MLIB)`, `mlib_ImageMedianFilterMxN_US(3MLIB)`,
- `mlib_ImageMinFilter3x3(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMinFilter5x5(3MLIB)`, `mlib_ImageMinFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMinFilter7x7(3MLIB)`, `mlib_ImageMinFilter7x7_Fp(3MLIB)`,
- `mlib_ImageRankFilter3x3(3MLIB)`, `mlib_ImageRankFilter3x3_US(3MLIB)`,
- `mlib_ImageRankFilter5x5(3MLIB)`, `mlib_ImageRankFilter5x5_Fp(3MLIB)`,
- `mlib_ImageRankFilter5x5_US(3MLIB)`, `mlib_ImageRankFilter3x3_US(3MLIB)`,
- `mlib_ImageRankFilter7x7(3MLIB)`, `mlib_ImageRankFilter7x7_Fp(3MLIB)`, `mlib_ImageRankFilter7x7_US(3MLIB)`,

---

562 man pages section 3: Multimedia Library Functions • Last Revised 2 Mar 2007
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageRankFilter3x3_Fp(3MLIB)

Name  
mlib_ImageRankFilter3x3_Fp – 3x3 rank filter

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRankFilter3x3_Fp(mlib_image *dst, 
    const mlib_image *src, mlib_s32 rank);

Description  
The mlib_ImageRankFilter3x3_Fp() function performs floating-point 3x3 rank filtering on 
an image. Each pixel of the destination image is the pixel with the user-specified rank in the 
filter window.

The source and destination images must be single-channel images.

Parameters  
The function takes the following arguments:

dst  Pointer to destination image.
src  Pointer to source image.
rank  The rank of the destination pixel. The pixel with minimum value is designated rank 0.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB),
mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB),
mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageRankFilter3x3_US()` function performs 3x3 rank filtering on an MLIB_SHORT type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window. The source and destination images must be single-channel images.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `rank` The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- `bits` The number of unsigned bits for pixel dynamic range. 9 ≤ bits ≤ 15.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also:

- `mlib_ImageMaxFilter3x3_US(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`, `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`, `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`, `mlib_ImageMedianFilter3x3_US(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`, `mlib_ImageMedianFilter5x5_US(3MLIB)`, `mlib_ImageMedianFilter5x5_Fp(3MLIB)`, `mlib_ImageMedianFilter7x7_US(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`, `mlib_ImageMinFilter3x3_US(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`, `mlib_ImageMinFilter5x5_US(3MLIB)`, `mlib_ImageMinFilter5x5_Fp(3MLIB)`, `mlib_ImageMinFilter7x7_US(3MLIB)`, `mlib_ImageMinFilter7x7_Fp(3MLIB)`, `mlib_ImageRankFilter3x3_US(3MLIB)`, `mlib_ImageRankFilter3x3_Fp(3MLIB)`.
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
Name
mlib_ImageRankFilter5x5 – 5x5 rank filter

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRankFilter5x5(mlib_image *dst, const mlib_image *src,
mlib_s32 rank);

Description
The mlib_ImageRankFilter5x5() function performs 5x5 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

Parameters
The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
rank The rank of the destination pixel. The pixel with minimum value is designated rank 0.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter5x5_US_Fp(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN(3MLIB),
mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter5x5_US(3MLIB), mlib_ImageMinFilter5x5_US_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter3x3_US_Fp(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US_Fp(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageRankFilter5x5_Fp()` function performs floating-point 5x5 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

The function takes the following arguments:

- `dst`: Pointer to destination image.
- `src`: Pointer to source image.
- `rank`: The rank of the destination pixel. The pixel with minimum value is designated rank 0.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),`
- `mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),`
- `mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),`
- `mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),`
- `mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),`
- `mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),`
- `mlib_ImageMinFilter5x5_US(3MLIB), mlib_ImageMinFilter5x5(3MLIB),`
- `mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),`
- `mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),`
- `mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),`
- `mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB),`
- `mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),`
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageRankFilter5x5_US()` function performs 5x5 rank filtering on an MLIB_SHORT type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

### Parameters
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `rank` The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- `bits` The number of unsigned bits for pixel dynamic range. 9 ≤ bits ≤ 15.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3_US(3MLIB)`, `mlib_ImageMedianFilter3x3_US_U(3MLIB)`,
- `mlib_ImageMedianFilter5x5(3MLIB)`, `mlib_ImageMedianFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMedianFilter5x5_US(3MLIB)`, `mlib_ImageMedianFilter5x5_US_U(3MLIB)`,
- `mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_US(3MLIB)`,
- `mlib_ImageMedianFilterMxN(3MLIB)`, `mlib_ImageMedianFilterMxN_Fp(3MLIB)`,
- `mlib_ImageMedianFilterMxN_US(3MLIB)`, `mlib_ImageMedianFilterMxN_US_U(3MLIB)`,
- `mlib_ImageMinFilter3x3(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMinFilter5x5(3MLIB)`, `mlib_ImageMinFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMinFilter7x7(3MLIB)`, `mlib_ImageMinFilter7x7_Fp(3MLIB)`,
- `mlib_ImageRankFilter3x3(3MLIB)`, `mlib_ImageRankFilter3x3_Fp(3MLIB)`,
- `mlib_ImageRankFilter5x5(3MLIB)`, `mlib_ImageRankFilter5x5_Fp(3MLIB)`,
- `mlib_ImageRankFilter3x3_US(3MLIB)`, `mlib_ImageRankFilter3x3_US_Fp(3MLIB)`,
- `mlib_ImageRankFilter5x5_US(3MLIB)`, `mlib_ImageRankFilter5x5_US_Fp(3MLIB)`.
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter7x7(3MLIB),
mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageRankFilter7x7(3MLIB)

Name  mlib_ImageRankFilter7x7 – 7x7 rank filter

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRankFilter7x7(mlib_image *dst, const mlib_image *src,
mlib_s32 rank);

Description  The mlib_ImageRankFilter7x7() function performs 7x7 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

Parameters  The function takes the following arguments:

dst      Pointer to destination image.
src      Pointer to source image.
rank      The rank of the destination pixel. The pixel with minimum value is designated rank 0.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB),
mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter5x5_US_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter3x3_US_Fp(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter5x5_US_Fp(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilter7x7_US_Fp(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageRankFilter7x7_Fp()` function performs floating-point 7x7 rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

### Parameters

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `rank` The rank of the destination pixel. The pixel with minimum value is designated rank 0.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB), mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB), mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB), mlib_ImageMaxFilter3x3_US(3MLIB), mlib_ImageMaxFilter5x5_US(3MLIB), mlib_ImageMaxFilter7x7_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageMinFilter3x3_US(3MLIB), mlib_ImageMinFilter5x5_US(3MLIB), mlib_ImageMinFilter7x7_US(3MLIB), mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB).
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_US(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageRankFilter7x7_US (MLIB)

# Name
mlib_ImageRankFilter7x7_US – 7x7 rank filter, unsigned

# Synopsis

c{ [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_ImageRankFilter7x7_US(mlib_image *dst,
                                    const mlib_image *src,
                                    mlib_s32 rank,
                                    mlib_s32 bits);

# Description
The mlib_ImageRankFilter7x7_US() function performs 7x7 rank filtering on an MLIB_SHORT type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

# Parameters
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **rank** The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- **bits** The number of unsigned bits for pixel dynamic range. \(9 \leq \text{bits} \leq 15\).

# Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

# Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

# See Also
mlib_ImageMaxFilter3x3(MLIB), mlib_ImageMaxFilter3x3_Fp(MLIB),
mlib_ImageMaxFilter5x5(MLIB), mlib_ImageMaxFilter5x5_Fp(MLIB),
mlib_ImageMaxFilter7x7(MLIB), mlib_ImageMaxFilter7x7_Fp(MLIB),
mlib_ImageMedianFilter3x3(MLIB), mlib_ImageMedianFilter3x3_Fp(MLIB),
mlib_ImageMedianFilter3x3_US(MLIB), mlib_ImageMedianFilter5x5(MLIB),
mlib_ImageMedianFilter5x5_US(MLIB), mlib_ImageMedianFilter7x7(MLIB),
mlib_ImageMedianFilter7x7_US(MLIB), mlib_ImageMedianFilterMxN(MLIB),
mlib_ImageMedianFilterMxN_Fp(MLIB), mlib_ImageMedianFilterMxN_US(MLIB),
mlib_ImageMinFilter3x3(MLIB), mlib_ImageMinFilter3x3_Fp(MLIB),
mlib_ImageMinFilter5x5(MLIB), mlib_ImageMinFilter5x5_Fp(MLIB),
mlib_ImageMinFilter7x7(MLIB), mlib_ImageMinFilter7x7_Fp(MLIB),
mlib_ImageRankFilter3x3(MLIB), mlib_ImageRankFilter3x3_Fp(MLIB),
mlib_ImageRankFilter3x3_US(MLIB), mlib_ImageRankFilter5x5(MLIB),
mlib_ImageRankFilter5x5_US(MLIB), mlib_ImageRankFilterMxN(MLIB),
mlib_ImageRankFilterMxN_Fp(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilterMxN_US(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_IMAGE_MAX_FILTER_MXN(MLIB)

mlib_ImageMaxFilter3x3(MLIB), mlib_ImageMaxFilter3x3_Fp(MLIB),
mlib_ImageMaxFilter5x5(MLIB), mlib_ImageMaxFilter5x5_Fp(MLIB),
mlib_ImageMaxFilter7x7(MLIB), mlib_ImageMaxFilter7x7_Fp(MLIB),
mlib_ImageMedianFilter3x3(MLIB), mlib_ImageMedianFilter3x3_Fp(MLIB),
mlib_ImageMedianFilter3x3_US(MLIB), mlib_ImageMedianFilter5x5(MLIB),
mlib_ImageMedianFilter5x5_Fp(MLIB), mlib_ImageMedianFilter5x5_US(MLIB),
mlib_ImageMedianFilter7x7(MLIB), mlib_ImageMedianFilter7x7_US(MLIB),
mlib_ImageMedianFilterMxN(MLIB), mlib_ImageMedianFilterMxN_Fp(MLIB),
mlib_ImageMedianFilterMxN_US(MLIB), mlib_ImageMinFilter3x3(MLIB),
mlib_ImageMinFilter3x3_Fp(MLIB), mlib_ImageMinFilter3x3_US(MLIB),
mlib_ImageMinFilter5x5(MLIB), mlib_ImageMinFilter5x5_Fp(MLIB),
mlib_ImageMinFilter5x5_US(MLIB), mlib_ImageMinFilter7x7(MLIB),
mlib_ImageMinFilter7x7_Fp(MLIB), mlib_ImageMinFilter7x7_US(MLIB),
mlib_ImageRankFilter3x3(MLIB), mlib_ImageRankFilter3x3_Fp(MLIB),
mlib_ImageRankFilter3x3_US(MLIB), mlib_ImageRankFilter5x5(MLIB),
mlib_ImageRankFilter5x5_US(MLIB), mlib_ImageRankFilterMxN(MLIB),
mlib_ImageRankFilterMxN_Fp(MLIB), mlib_ImageRankFilterMxN_US(MLIB),
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageRankFilterMxN()` function performs MxN rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

### Parameters

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

- **m**  
  Width of the filter window. `m` must be odd number greater than 1.

- **n**  
  Height of the filter window. `n` must be odd number greater than 1.

- **rank**  
  The rank of the destination pixel. The pixel with minimum value is designated rank 0.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB), mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB), mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB), mlib_ImageMedianFilter3x3_US(3MLIB), mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5_Fp(3MLIB), mlib_ImageMedianFilter5x5_US(3MLIB), mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB), mlib_ImageMedianFilter7x7_US(3MLIB), mlib_ImageMedianFilterMxN_Fp(3MLIB), mlib_ImageMedianFilterMxN_US(3MLIB), mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB), mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB), mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB), mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB), mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB), mlib_ImageRankFilterMxN_US(3MLIB)`. 
mlib_ImageRankFilterMxN(3MLIB)

mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
mlib_ImageRankFilterMxN_Fp(3MLIB)

Name

mlib_ImageRankFilterMxN_Fp – MxN rank filter

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRankFilterMxN_Fp(mlib_image *dst,
const mlib_image *src, mlib_s32 m, mlib_s32 n, mlib_s32 rank);

Description

The mlib_ImageRankFilterMxN_Fp() function performs floating-point MxN rank filtering on an image. Each pixel of the destination image is the pixel with the user-specified rank in the filter window.

The source and destination images must be single-channel images.

Parameters

The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
m Width of the filter window. m must be odd number greater than 1.
n Height of the filter window. n must be odd number greater than 1.
rank The rank of the destination pixel. The pixel with minimum value is designated rank 0.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter3x3(3MLIB), mlib_ImageMedianFilter3x3_Fp(3MLIB),
mlib_ImageMedianFilter5x5(3MLIB), mlib_ImageMedianFilter5x5(3MLIB),
mlib_ImageMedianFilter5x5_UTM(3MLIB), mlib_ImageMedianFilter5x5_UTM_Fp(3MLIB),
mlib_ImageMedianFilter7x7(3MLIB), mlib_ImageMedianFilter7x7_Fp(3MLIB),
mlib_ImageMedianFilter7x7_UTM(3MLIB), mlib_ImageMedianFilter7x7_UTM_Fp(3MLIB),
mlib_ImageMinFilter3x3(3MLIB), mlib_ImageMinFilter3x3_Fp(3MLIB),
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageMinFilter7x7_UTM(3MLIB), mlib_ImageMinFilter7x7_UTM_Fp(3MLIB),
mlib_ImageMinFilterMxN(3MLIB), mlib_ImageMinFilterMxN_Fp(3MLIB),
mlib_ImageMinFilterMxN_UTM(3MLIB), mlib_ImageMinFilterMxN_UTM_Fp(3MLIB),
mlib_ImageMaxFilter3x3(3MLIB), mlib_ImageMaxFilter3x3_Fp(3MLIB),
mlib_ImageMaxFilter5x5(3MLIB), mlib_ImageMaxFilter5x5_Fp(3MLIB),
mlib_ImageMaxFilter7x7(3MLIB), mlib_ImageMaxFilter7x7_Fp(3MLIB),
mlib_ImageMaxFilter7x7_UTM(3MLIB), mlib_ImageMaxFilter7x7_UTM_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter5x5(3MLIB), mlib_ImageRankFilter5x5_Fp(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_UTM(3MLIB), mlib_ImageRankFilter7x7_UTM_Fp(3MLIB),
mlib_ImageRankFilterMxN(3MLIB), mlib_ImageRankFilterMxN_Fp(3MLIB),
mlib_ImageRankFilterMxN_UTM(3MLIB), mlib_ImageRankFilterMxN_UTM_Fp(3MLIB),
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB),
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB),
mlib_ImageRankFilterMxN_US(3MLIB), attributes(5)
The `mlib_ImageRankFilterMxN_US()` function performs \( M \times N \) rank filtering on an MLIB\_SHORT type of image that contains unsigned data. Each pixel of the destination image is the pixel with the user-specified rank in the filter window. The source and destination images must be single-channel images.

### Parameters

- **dst**  
  Pointer to destination image.
- **src**  
  Pointer to source image.
- **m**  
  Width of the filter window. \( m \) must be an odd number greater than \( 1 \).
- **n**  
  Height of the filter window. \( n \) must be an odd number greater than \( 1 \).
- **rank**  
  The rank of the destination pixel. The pixel with minimum value is designated rank 0.
- **bits**  
  The number of unsigned bits for pixel dynamic range. \( 9 \leq \text{bits} \leq 15 \).

### Return Values

The function returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageMaxFilter3x3(3MLIB)`, `mlib_ImageMaxFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMaxFilter5x5(3MLIB)`, `mlib_ImageMaxFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMaxFilter7x7(3MLIB)`, `mlib_ImageMaxFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter3x3(3MLIB)`, `mlib_ImageMedianFilter3x3_Fp(3MLIB)`,
- `mlib_ImageMedianFilter5x5(3MLIB)`, `mlib_ImageMedianFilter5x5_Fp(3MLIB)`,
- `mlib_ImageMedianFilter5x5_US(3MLIB)`, `mlib_ImageMedianFilter5x5_US_Fp(3MLIB)`,
- `mlib_ImageMedianFilter7x7(3MLIB)`, `mlib_ImageMedianFilter7x7_Fp(3MLIB)`,
- `mlib_ImageMedianFilter7x7_US(3MLIB)`, `mlib_ImageMedianFilter7x7_US_Fp(3MLIB)`,
- `mlib_ImageMedianFilterMxN_Fp(3MLIB)`, `mlib_ImageMedianFilterMxN_US(3MLIB)`,
- `mlib_ImageMinFilter3x3(3MLIB)`, `mlib_ImageMinFilter3x3_Fp(3MLIB)`,
mlib_ImageMinFilter5x5(3MLIB), mlib_ImageMinFilter5x5_Fp(3MLIB),
mlib_ImageMinFilter7x7(3MLIB), mlib_ImageMinFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter3x3(3MLIB), mlib_ImageRankFilter3x3_Fp(3MLIB),
mlib_ImageRankFilter3x3_US(3MLIB), mlib_ImageRankFilter5x5(3MLIB),
mlib_ImageRankFilter5x5_Fp(3MLIB), mlib_ImageRankFilter5x5_US(3MLIB),
mlib_ImageRankFilter7x7(3MLIB), mlib_ImageRankFilter7x7_Fp(3MLIB),
mlib_ImageRankFilter7x7_US(3MLIB), mlib_ImageRankFilterMxN(3MLIB),
mlib_ImageRankFilterMxN_Fp(3MLIB), attributes(5)
The `mlib_ImageReformat()` function copies and casts, if needed, an image from one buffer to another. The formats and data types of the two buffers may be different.

\[
\text{dstPixel}[x][y][i] = (\text{dstDataType}) \text{ srcPixel}[x][y][i]
\]

where the values of a pixel at position \((x, y)\) and in channel \(i\) are:

\[
\text{srcPixel}[x][y][i] = \text{srcData}[i][\text{srcBandoffsets}[i] + \text{srcScanlinestride*y} + \text{srcPixelstride*x}]
\]

\[
\text{dstPixel}[x][y][i] = \text{dstData}[i][\text{dstBandoffsets}[i] + \text{dstScanlinestride*y} + \text{dstPixelstride*x}]
\]

It is the user’s responsibility to make sure that the data buffers supplied are suitable for this operation. The `srcData` and `dstData` can have 1, 2, 3, or 4 channels, and they must have the same number of channels. The `srcDataType` and `dstDataType` can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.

The conversions between different data types are implemented as described in the following table:

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Dest. Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MLIB_SHORT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)\text{clamp}(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_USHORT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)\text{clamp}(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_INT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)\text{clamp}(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_FLOAT</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)\text{clamp}(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_DOUBLE</code></td>
<td><code>MLIB_BYTE</code></td>
<td><code>(mlib_u8)\text{clamp}(x, 0, 255)</code></td>
</tr>
<tr>
<td><code>MLIB_BYTE</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(mlib_s16)x</code></td>
</tr>
<tr>
<td><code>MLIB_USHORT</code></td>
<td><code>MLIB_SHORT</code></td>
<td><code>(mlib_s16)\text{clamp}(x, -32768, 32767)</code></td>
</tr>
</tbody>
</table>
The actions are defined in C-style pseudo-code. All type casts follow the rules of standard C. `clamp()` can be defined as a macro:

```c
#define clamp(x, low, high) (((x) < (low)) ? (low) : (((x) > (high)) ? (high) : (x)))
```

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Dest. Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_INT</td>
<td>MLIB_SHORT</td>
<td>(mlib_s16)clamp(x, -32768, 32767)</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_SHORT</td>
<td>(mlib_s16)clamp(x, -32768, 32767)</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_SHORT</td>
<td>(mlib_s16)clamp(x, -32768, 32767)</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_USHORT</td>
<td>(mlib_u16)clamp(x, 0, 65535)</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_INT</td>
<td>(mlib_s32)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_INT</td>
<td>(mlib_s32)x</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>MLIB_INT</td>
<td>(mlib_s32)x</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_INT</td>
<td>(mlib_s32)clamp(x, -2147483647-1, 2147483647)</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_INT</td>
<td>(mlib_s32)clamp(x, -2147483647-1, 2147483647)</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>MLIB_FLOAT</td>
<td>(mlib_f32)x</td>
</tr>
<tr>
<td>MLIB_BYTE</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
<tr>
<td>MLIB_FLOAT</td>
<td>MLIB_DOUBLE</td>
<td>(mlib_d64)x</td>
</tr>
</tbody>
</table>
The function takes the following arguments:

- **dstData** The pointer to the destination image data buffer.
- **srcData** The pointer to the source image data buffer.
- **numBands** The number of channels of the image data buffers.
- **xSize** The width of the image.
- **ySize** The height of the image.
- **dstDataType** The data type of the `dstData` buffer.
- **dstBandoffsets** The initial pixel's offsets in the `dstData` buffer in terms of destination data buffer elements.
- **dstScanlinestride** The scanline stride of the `dstData` buffer in terms of destination data buffer elements.
- **dstPixelstride** The pixel stride of the `dstData` buffer in terms of destination data buffer elements.
- **srcDataType** The data type of the `srcData` buffer.
- **srcBandoffsets** The initial pixel's offsets in the `srcData` buffer in terms of source data buffer elements.
- **srcScanlinestride** The scanline stride of the `srcData` buffer in terms of source data buffer elements.
- **srcPixelstride** The pixel stride of the `srcData` buffer in terms of source data buffer elements.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_ImageDataTypeConvert(3MLIB), attributes(5)
**Name**  
mlib_ImageReplaceColor – replace a color in an image

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_ImageReplaceColor(mlib_image *dst, const mlib_image *src,  
        const mlib_s32 *color1, const mlib_s32 *color2);
```

**Description**  
The `mlib_ImageReplaceColor()` function copies the source image to the destination image and replaces the pixels having a value of `color1` with `color2`.

It uses the following equation:

\[
\begin{align*}
    \text{dst}[x][y] &= \text{color2} & \text{if src}[x][y] &= \text{color1} \\
    \text{dst}[x][y] &= \text{src}[x][y] & \text{if src}[x][y] &\neq \text{color1}
\end{align*}
\]

**Parameters**  
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **color1** Array of color components to be replaced.
- **color2** Array of color components to replace `color1`.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageReplaceColor_Inp(3MLIB), mlib_ImageReplaceColor_Fp(3MLIB),  
mlib_ImageReplaceColor_Fp_Inp(3MLIB), mlib_ImageThresh5(3MLIB),  
mlib_ImageThresh5_Inp(3MLIB), mlib_ImageThresh5_Fp(3MLIB),  
mlib_ImageThresh5_Fp_Inp(3MLIB), attributes(5)
The `mlib_ImageReplaceColor_Fp()` function copies the source image to the destination image and replaces the pixels having a value of `color1` with `color2`.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y] &= \text{color2} \quad \text{if} \quad \text{src}[x][y] == \text{color1} \\
\text{dst}[x][y] &= \text{src}[x][y] \quad \text{if} \quad \text{src}[x][y] \neq \text{color1}
\end{align*}
\]

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `color1` Array of color components to be replaced.
- `color2` Array of color components to replace `color1`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See Also `mlib_ImageReplaceColor(3MLIB), mlib_ImageReplaceColor_Inp(3MLIB), mlib_ImageReplaceColor_Fp_Inp(3MLIB), mlib_ImageReplaceColor_Fp_Fp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Inp(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), attributes(5)`
The `mlib_ImageReplaceColor_Fp_Inp()` function scans the image for all pixels with color value equal to `color1` and replaces these pixels with `color2`.

It uses the following equation:

\[ \text{srcdst}[x][y] = \text{color2} \text{ if srcdst}[x][y] == \text{color1} \]

**Parameters**

- `srcdst` Pointer to the source and destination image.
- `color1` Array of color components to be replaced.
- `color2` Array of color components to replace `color1`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_ImageReplaceColor(3MLIB), mlib_ImageReplaceColor_Inp(3MLIB), mlib_ImageReplaceColor_Fp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Inp(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), attributes(5)`
mlib_ImageReplaceColor_Inp(3MLIB)

Name mlib_ImageReplaceColor_Inp – replace a color in an image, in place

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageReplaceColor_Inp(mlib_image *srcdst,
const mlib_s32 *color1, const mlib_s32 *color2);

Description The mlib_ImageReplaceColor_Inp() function scans the image for all pixels with color value equal to color1 and replaces these pixels with color2.

It uses the following equation:
srcdst[x][y] = color2  if srcdst[x][y] == color1

Parameters The function takes the following arguments:
srcdst Pointer to the source and destination image.
color1 Array of color components to be replaced.
color2 Array of color components to replace color1.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageReplaceColor(3MLIB), mlib_ImageReplaceColor_Fp(3MLIB),
mlib_ImageReplaceColor_Fp_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), mlib_ImageThresh5_Fp(3MLIB),
mlib_ImageThresh5_Fp_Inp(3MLIB), attributes(5)
mlib_ImageResetStruct (3MLIB)

Name
mlib_ImageResetStruct – reset image data structure

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageResetStruct(mlib_image *image, mlib_type type,
       mlib_s32 channels, mlib_s32 width, mlib_s32 height, mlib_s32 stride,
       const void *datbuf);

Description
The mlib_ImageResetStruct() function resets a mediaLib image data structure using
parameters supplied by the user.

The mlib_ImageResetStruct() function returns MLIB_FAILURE if the supplied parameters do
not pass the following sanity checks:

- image should not be NULL
- type should be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT,
  or MLIB_DOUBLE
- channels should be between 1 and 4
- width should be greater than 0
- height should be greater than 0
- stride should be no less than width * channels * (size of type in bytes)

Whenever MLIB_FAILURE is returned, the original image data structure is not changed.

Whenever datbuf is NULL, the original data buffer is reused. If
mlib_ImageIsUserAllocated(image)==0, such as the case the image data structure was
created by mlib_ImageCreate(), and the data buffer size required by the parameters supplied
is larger than the original, MLIB_FAILURE is returned.

Whenever datbuf is not NULL, if mlib_ImageIsUserAllocated(image)==0, the original data
buffer is freed, otherwise the original data buffer is not freed. If datbuf points to the original
data buffer, it is not freed.

Parameters
The function takes the following arguments:

- image Pointer to the image data structure.
- type Image data type. It can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT,
  MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.
- channels Number of channels in the image.
- width Width of image in pixels.
- height Height of image in pixels.
- stride Stride of each row of the data space in bytes.
- datbuf Pointer to the image data buffer.
Return Values  MLIB_SUCCESS is returned if the image data structure is reset successfully. MLIB_FAILURE is returned when the image data structure cannot be reset according to the parameters supplied.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageCreate(3MLIB), mlib_ImageCreateSubimage(3MLIB),
          mlib_ImageCreateStruct(3MLIB), mlib_ImageSetStruct(3MLIB),
          mlib_ImageDelete(3MLIB), mlib_ImageSetFormat(3MLIB),
          mlib_ImageSetPaddings(3MLIB), attributes(5)
mlib_ImageResetSubimageStruct

**Name**
mlib_ImageResetSubimageStruct – reset sub-image data structure

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_ImageResetSubimageStruct(mlib_image *subimg,
                                          const mlib_image *img, mlib_s32 x, mlib_s32 y,
                                          mlib_s32 w, mlib_s32 h);
```

**Description**
The `mlib_ImageResetSubimageStruct()` function resets a sub-image's data structure using parameters supplied by the user.

The `mlib_ImageResetSubimageStruct()` function returns `MLIB_FAILURE` if the supplied parameters do not pass the following sanity checks:

- `subimg` != NULL
- `img` != NULL
- `0 < w <= mlib_ImageGetWidth(img)`
- `0 < h <= mlib_ImageGetHeight(img)`
- `0 <= x <= (mlib_ImageGetWidth(img) - w)`
- `0 <= y <= (mlib_ImageGetHeight(img) - h)`

Whenever `MLIB_FAILURE` is returned, the original image data structure is not changed.

If `mlib_ImageIsUserAllocated(subimg)==0`, the original data buffer is freed, otherwise the original data buffer is not freed.

**Parameters**
The function takes the following arguments:

- `subimg` Pointer to the sub-image data structure.
- `img` Pointer to the source image data structure.
- `x` X coordinate of the left border in the source image.
- `y` Y coordinate of the top border in the source image.
- `w` Width of the sub-image.
- `h` Height of the sub-image.

**Return Values**
`MLIB_SUCCESS` is returned if the image data structure is reset successfully. `MLIB_FAILURE` is returned when the image data structure cannot be reset according to the parameters supplied.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageCreate(3MLIB), mlib_ImageCreateSubimage(3MLIB),
mlib_ImageCreateStruct(3MLIB), mlib_ImageSetStruct(3MLIB),
mlib_ImageResetStruct(3MLIB), mlib_ImageSetSubimageStruct(3MLIB),
mlib_ImageDelete(3MLIB), mlib_ImageSetFormat(3MLIB),
mlib_ImageSetPaddings(3MLIB), attributes(5)
Name
mlib_ImageRotate180 – rotate an image by 180 degrees

Synopsis
cc [ flag...] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRotate180(mlib_image *dst, const mlib_image *src);

Description
Rotate an image 180 degrees.

The width and height of the destination image can be different from the width and height of
the source image. The center of the source image is mapped to the center of the destination
image.

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or
MLIB_INT.

Parameters
The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB),
mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90(3MLIB),
mlib_ImageRotate90_Fp(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
mlib_ImageRotate180_Fp (3MLIB)

Name  mlib_ImageRotate180_Fp – rotate an image by 180 degrees

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRotate180_Fp(mlib_image *dst, const mlib_image *src);

Description  Rotate a floating-point image 180 degrees.

The width and height of the destination image can be different from the width and height of
the source image. The center of the source image is mapped to the center of the destination
image.

Parameters  The function takes the following arguments:
  dst  Pointer to destination image.
  src  Pointer to source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB),
mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90(3MLIB),
mlib_ImageRotate90_Fp(3MLIB), mlib_ImageRotate180(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
**Name**
mlib_ImageRotate270 - rotate an image by 270 degrees

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRotate270(mlib_image *dst, const mlib_image *src);

**Description**
Rotate an image 270 degrees clockwise.

The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.

The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.

**Parameters**
The function takes the following arguments:
- **dst** Pointer to destination image.
- **src** Pointer to source image.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB),
mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90(3MLIB),
mlib_ImageRotate90_Fp(3MLIB), mlib_ImageRotate180(3MLIB),
mlib_ImageRotate180_Fp(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
Name  mlib_ImageRotate270_Fp -- rotate an image by 270 degrees

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>
          
          mlib_status mlib_ImageRotate270_Fp(mlib_image *dst, const mlib_image *src);

Description  Rotate a floating-point image 270 degrees clockwise.

          The width and height of the destination image can be different from the width and height of
          the source image. The center of the source image is mapped to the center of the destination
          image.

Parameters  The function takes the following arguments:
          
          dst    Pointer to destination image.
          src    Pointer to source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

          +-----------------------------------+---------------------+
          | ATTRIBUTE TYPE | ATTRIBUTE VALUE      |
          +----------------+---------------------+
          | Interface Stability | Committed            |
          | MT-Level          | MT-Safe              |
          +-------------------+---------------------+

See Also  mlib_ImageFlipAntiDiag(3MLIB),mlib_ImageFlipAntiDiag_Fp(3MLIB),
          mlib_ImageFlipMainDiag(3MLIB),mlib_ImageFlipMainDiag_Fp(3MLIB),
          mlib_ImageFlipX(3MLIB),mlib_ImageFlipX_Fp(3MLIB),mlib_ImageFlipY(3MLIB),
          mlib_ImageFlipY_Fp(3MLIB),mlib_ImageRotate90(3MLIB),
          mlib_ImageRotate90_Fp(3MLIB),mlib_ImageRotate180(3MLIB),
          mlib_ImageRotate180_Fp(3MLIB),mlib_ImageRotate270(3MLIB),attributes(5)
### mlib_ImageRotate(3MLIB)

<table>
<thead>
<tr>
<th>Name</th>
<th>mlib_ImageRotate - rotate image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td></td>
</tr>
</tbody>
</table>

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRotate(mlib_image *dst, const mlib_image *src,
                              mlib_d64 angle, mlib_d64 xcenter, mlib_d64 ycenter, mlib_filter filter,
                              mlib_edge edge);
```

### Description
The `mlib_ImageRotate()` function rotates a source image around a user-defined rotation center in the user-defined radians.

The width and height of the destination image can be different from the width and height of the source image. The \((xcenter, ycenter)\) point of the source image is mapped to the center of the destination image. You should ensure that the destination buffer is large enough to hold the resulting bounding box to avoid clipping part of the image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

### Parameters
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `angle` Angle of rotation. The angle is measured in radians clockwise.
- `xcenter` X coordinate of rotation center in the source image.
- `ycenter` Y coordinate of rotation center in the source image.
- `filter` Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageRotate_Fp(3MLIB), mlib_ImageRotateIndex(3MLIB), attributes(5)
mlib_ImageRotate90(3MLIB)

<table>
<thead>
<tr>
<th>Name</th>
<th>mlib_ImageRotate90 – rotate an image by 90 degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>cc [ flag... ] file... -lmlib [ library... ]</td>
</tr>
<tr>
<td></td>
<td>#include &lt;mlib.h&gt;</td>
</tr>
<tr>
<td></td>
<td>mlib_status mlib_ImageRotate90(mlib_image *dst, const mlib_image *src);</td>
</tr>
<tr>
<td>Description</td>
<td>Rotate an image 90 degrees clockwise.</td>
</tr>
<tr>
<td></td>
<td>The width and height of the destination image can be different from the width and height of the source image. The center of the source image is mapped to the center of the destination image.</td>
</tr>
<tr>
<td></td>
<td>The data type of the images can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, or MLIB_INT.</td>
</tr>
<tr>
<td>Parameters</td>
<td>The function takes the following arguments:</td>
</tr>
<tr>
<td></td>
<td>dst     Pointer to destination image.</td>
</tr>
<tr>
<td></td>
<td>src     Pointer to source image.</td>
</tr>
<tr>
<td>Return Values</td>
<td>The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.</td>
</tr>
<tr>
<td>Attributes</td>
<td>See attributes(5) for descriptions of the following attributes:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB),
mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90_Fp(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
Name  mlib_ImageRotate90_Fp – rotate an image by 90 degrees

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRotate90_Fp(mlib_image *dst, const mlib_image *src);

Description  Rotate a floating-point image 90 degrees clockwise.

The width and height of the destination image can be different from the width and height of
the source image. The center of the source image is mapped to the center of the destination
image.

Parameters  The function takes the following arguments:

dst  Pointer to destination image.

src  Pointer to source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageFlipAntiDiag(3MLIB), mlib_ImageFlipAntiDiag_Fp(3MLIB),
mlib_ImageFlipMainDiag(3MLIB), mlib_ImageFlipMainDiag_Fp(3MLIB),
mlib_ImageFlipX(3MLIB), mlib_ImageFlipX_Fp(3MLIB), mlib_ImageFlipY(3MLIB),
mlib_ImageFlipY_Fp(3MLIB), mlib_ImageRotate90(3MLIB),
mlib_ImageRotate180(3MLIB), mlib_ImageRotate180_Fp(3MLIB),
mlib_ImageRotate270(3MLIB), mlib_ImageRotate270_Fp(3MLIB), attributes(5)
The `mlib_ImageRotate_Fp()` function rotates a floating-point source image around a user-defined rotation center in the user-defined radians.

The width and height of the destination image can be different from the width and height of the source image. The \((x_{center}, y_{center})\) point of the source image is mapped to the center of the destination image. You should ensure that the destination buffer is large enough to hold the resulting bounding box to avoid clipping part of the image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

### Parameters
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **angle** Angle of rotation. The angle is measured in radians clockwise.
- **xcenter** X coordinate of rotation center in the source image.
- **ycenter** Y coordinate of rotation center in the source image.
- **filter** Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:
mlib_ImageRotate_Fp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageRotate(3MLIB), mlib_ImageRotateIndex(3MLIB), attributes(5)
Name  
mlib_ImageRotateIndex – rotate color-indexed image

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageRotateIndex(mlib_image *dst, const mlib_image *src,
    mlib_d64 angle, mlib_d64 xcenter, mlib_d64 ycenter, mlib_filter filter,
    mlib_edge edge, const void *colormap);

Description  
The mlib_ImageRotateIndex() function rotates the source image about a user-defined rotation center in user-defined radians.

The width and height of the destination image can be different from the width and height of the source image. The (xcenter, ycenter) point of the source image is mapped to the center of the destination image. You should ensure that the destination buffer is large enough to hold the resulting bounding box to avoid clipping part of the image.

The source and destination images must be single-channel images.

The image data type must be MLIB_BYTE or MLIB_SHORT.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

Parameters  
The function takes the following arguments:

dst  Pointer to destination image.
src  Pointer to source image.
angle  Angle of rotation. The angle is measured in radians clockwise.
xcenter  X coordinate of rotation center in the source image.
ycenter  Y coordinate of rotation center in the source image.
filter  Type of resampling filter. It can be one of the following:

MLIB_NEAREST
MLIB_BILINEAR
MLIB_BICUBIC
MLIB_BICUBIC2

dedge  Type of edge condition. It can be one of the following:

MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_PADDED

colormap  Internal data structure for inverse color mapping. This data structure is generated by the mlib_ImageColorTrue2IndexInit() function.
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageRotate(3MLIB), mlib_ImageRotate_Fp(3MLIB), attributes(5)
The `mlib_ImageScalarBlend()` function blends the first and second source images by adding each of their scaled pixels. The first source image is scaled by the scalar \( a \), and the second source image is inverse scaled by \((1 - a)\).

It uses the following equation:

\[
dst[x][y][i] = a[i] * src1[x][y][i] + (1 - a[i]) * src2[x][y][i]
\]

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src1** Pointer to first source image.
- **src2** Pointer to second source image.
- **alpha** Scalar blending factor. The \( a \) value equals \((\alpha * 2^{-31})\). \( \alpha[i] \) contains the blending factor for channel \( i \).

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageScalarBlend_Fp(3MLIB)`, `mlib_ImageScalarBlend_Fp_Inp(3MLIB)`, `mlib_ImageScalarBlend_Inp(3MLIB)`, attributes(5)
mlib_ImageScalarBlend_Fp

**Name**  mlib_ImageScalarBlend_Fp – image blending with scalar

**Synopsis**  
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageScalarBlend_Fp(mlib_image *dst,
        const mlib_image *src1, const mlib_image *src2, const mlib_d64 *alpha);
```

**Description**  The `mlib_ImageScalarBlend_Fp()` function blends the first and second floating-point source images by adding each of their scaled pixels. The first source image is scaled by the scalar `a`, and the second source image is inverse scaled by `(1 - a)`.

It uses the following equation:

```
dst[x][y][i] = a[i]*src1[x][y][i] + (1 - a[i])*src2[x][y][i]
```

**Parameters**  The function takes the following arguments:

- `dst`  Pointer to destination image.
- `src1`  Pointer to first source image.
- `src2`  Pointer to second source image.
- `alpha`  Scalar blending factor. The `a` value equals `alpha` which should be in the `[0.0, 1.0]` range. `alpha[i]` contains the blending factor for channel `i`.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_ImageBlend(3MLIB)`, `mlib_ImageScalarBlend_Fp_Inp(3MLIB)`, `mlib_ImageScalarBlend_Inp(3MLIB)`, `attributes(5)`
mlib_ImageScalarBlend_Fp_Inp(3MLIB)

Name  mlib_ImageScalarBlend_Fp_Inp – image blending with scalar

Synopsis  cc [ flag ... ] file ... -lmlib [ library ... ]
#include <mlib.h>

mlib_status mlib_ImageScalarBlend_Fp_Inp(mlib_image *src1dst,
const mlib_image *src2, const mlib_d64 *alpha);

Description  The mlib_ImageScalarBlend_Fp_Inp() function blends the first and second floating-point source images by adding each of their scaled pixels in place. The first source image is scaled by the scalar $a$, and the second source image is inverse scaled by $(1 - a)$.

It uses the following equation:

$$
src1dst[x][y][i] = a[i]*src1dst[x][y][i] + (1 - a[i])*src2[x][y][i]
$$

Parameters  The function takes the following arguments:

- **src1dst**: Pointer to first source and destination image.
- **src2**: Pointer to second source image.
- **alpha**: Scalar blending factor. The $a$ value equals $alpha$ which should be in the $[0.0, 1.0]$ range. $alpha[i]$ contains the blending factor for channel $i$.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageScalarBlend(3MLIB), mlib_ImageScalarBlend_Fp(3MLIB), mlib_ImageScalarBlend_Inp(3MLIB), attributes(5)
The `mlib_ImageScalarBlend_Inp()` function blends the first and second source images by adding each of their scaled pixels in place. The first source image is scaled by the scalar `a`, and the second source image is inverse scaled by `(1 - a)`.

It uses the following equation:

```
srcldst[x][y][i] = a[i]*srcldst[x][y][i] + 
                   (1 - a[i])*src2[x][y][i]
```

**Parameters**

The function takes the following arguments:

- `srcldst` Pointer to first source and destination image.
- `src2` Pointer to second source image.
- `alpha` Scalar blending factor. The `a` value equals `(alpha * 2**(-31)).` `alpha[i]` contains the blending factor for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageScalarBlend(3MLIB)`, `mlib_ImageScalarBlend_Fp(3MLIB)`, `mlib_ImageScalarBlend_Fp_Inp(3MLIB)`, attributes(5)
# mlib_ImageScale2

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageScale2(mlib_image *dst, const mlib_image *src, const mlib_d64 *alpha, const mlib_d64 *beta);
```

## Description

The `mlib_ImageScale2()` function performs a linear scaling on the pixels of the source image by multiplying the data by a scale factor and then adding an offset. Images must have the same size, and number of channels. They can have 1, 2, 3, or 4 channels.

The following equation is used:

\[
dst[x][y][i] = src[x][y][i] \times \alpha[i] + \beta[i]
\]

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

See the following table for available variations of this linear scaling function.

<table>
<thead>
<tr>
<th>Type [*]</th>
<th>BYTE</th>
<th>SHORT</th>
<th>USHORT</th>
<th>INT</th>
<th>FLOAT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BYTE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[*] Each row represents a source data type. Each column represents a destination data type.

## Parameters

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **alpha**: Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- **beta**: Offset value. `beta[i]` contains the offset for channel `i`.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:
mlib_ImageScale2(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageScale(3MLIB), mlib_ImageScale_Fp(3MLIB),  
mlib_ImageScale_Fp_Inp(3MLIB), mlib_ImageScale_Inp(3MLIB),  
mlib_ImageScale2_Inp(3MLIB), attributes(5)
# mlib_ImageScale2_Inp

## Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageScale2_Inp(mlib_image *srcdst, const mlib_d64 *alpha,
                      const mlib_d64 *beta);
```

## Description

The `mlib_ImageScale2_Inp()` function performs an in-place linear scaling on the pixels of the source image by multiplying the data by a scale factor and then adding an offset. Images can have 1, 2, 3, or 4 channels.

The following equation is used:

```
srcdst[x][y][i] = srcdst[x][y][i] * alpha[i] + beta[i]
```

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

The image can be of type `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT` or `MLIB_INT`.

## Parameters

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `alpha` Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.  
- `beta` Offset value. `beta[i]` contains the offset for channel `i`.  

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_ImageScale(3MLIB)`, `mlib_ImageScale_Fp(3MLIB)`, `mlib_ImageScale_Fp_Inp(3MLIB)`, `mlib_ImageScale_Inp(3MLIB)`, `mlib_ImageScale2(3MLIB)`, `attributes(5)`
The `mlib_ImageScale()` function performs a linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

\[ \text{dst}[x][y][i] = \text{src}[x][y][i] \times \text{alpha}[i] \times 2^{-\text{shift}} + \text{beta}[i] \]

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

See the following table for available variations of this linear scaling function.

<table>
<thead>
<tr>
<th>Type</th>
<th>BYTE</th>
<th>SHORT</th>
<th>USHORT</th>
<th>INT</th>
<th>FLOAT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_BYTE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_SHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_USHORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_INT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[*] Each row represents a source data type. Each column represents a destination data type.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `alpha` Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- `beta` Offset value. `beta[i]` contains the offset for channel `i`.
- `shift` Right shifting factor.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`. 
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageScale_Fp(3MLIB), mlib_ImageScale_Fp_Inp(3MLIB),
          mlib_ImageScale_Inp(3MLIB), mlib_ImageScale2(3MLIB),
          mlib_ImageScale2_Inp(3MLIB), attributes(5)
#include <mlib.h>

mlib_status mlib_ImageScale_Fp(mlib_image *dst, const mlib_image *src, const mlib_d64 *alpha, const mlib_d64 *beta);

The `mlib_ImageScale_Fp()` function performs a floating-point linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

\[
\text{dst}[x][y][i] = \text{src}[x][y][i] \times \alpha[i] + \beta[i]
\]

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

See the following table for available variations of this linear scaling function.

<table>
<thead>
<tr>
<th>Type[*]</th>
<th>BYTE</th>
<th>SHORT</th>
<th>USHORT</th>
<th>INT</th>
<th>FLOAT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLIB_FLOAT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MLIB_DOUBLE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[*] Each row represents a source data type. Each column represents a destination data type.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `alpha` Scaling factor. \(\alpha[i]\) contains the scaling factor for channel \(i\).
- `beta` Offset value. \(\beta[i]\) contains the offset for channel \(i\).

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageScale(3MLIB), mlib_ImageScale_Fp_Inp(3MLIB),
mlib_ImageScale_Inp(3MLIB), mlib_ImageScale2(3MLIB),
mlib_ImageScale2_Inp(3MLIB), attributes(5)
The `mlib_ImageScale_Fp_Inp()` function performs a floating-point, in-place linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

\[
\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] \times \text{alpha}[i] + \text{beta}[i]
\]

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

The image can be of type `MLIB_FLOAT` or `MLIB_DOUBLE`.

**Parameters**

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `alpha` Scaling factor. `alpha[i]` contains the scaling factor for channel `i`.
- `beta` Offset value. `beta[i]` contains the offset for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageScale(3MLIB), mlib_ImageScale_Fp(3MLIB), mlib_ImageScale_Inp(3MLIB), mlib_ImageScale2(3MLIB), mlib_ImageScale2_Inp(3MLIB), attributes(5)`
**Name**    mlib_ImageScale_Inp – linear scaling, in place

**Synopsis**    cc [ flag... ] file... -lmlib [ library... ]
                #include <mlib.h>

                mlib_status mlib_ImageScale_Inp(mlib_image *srcdst, const mlib_s32 *alpha,
                                               const mlib_s32 *beta, mlib_s32 shift);

**Description**    The mlib_ImageScale_Inp() function performs an in-place linear scaling on the pixels of the source image by multiplying the data by a scale factor, shifting, and then adding an offset.

The following equation is used:

\[
\text{srcdst}[x][y][i] = \text{srcdst}[x][y][i] \times \alpha[i] \times 2^{(-\text{shift})} + \beta[i]
\]

If the result of the operation underflows/overflows the minimum/maximum value supported by the destination image, then it will be clamped to the minimum/maximum value respectively.

The image can be of type MLIB_BYTE, MLIB_SHORT, MLIB_USHORT or MLIB_INT.

**Parameters**    The function takes the following arguments:

* srcdst    Pointer to source and destination image.
* alpha    Scaling factor. alpha[i] contains the scaling factor for channel i.
* beta    Offset value. beta[i] contains the offset for channel i.
* shift    Right shifting factor.

**Return Values**    The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**    See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**    mlib_ImageScale(3MLIB), mlib_ImageScale_Fp(3MLIB),
                mlib_ImageScale_Fp_Inp(3MLIB), mlib_ImageScale2(3MLIB),
                mlib_ImageScale2_Inp(3MLIB), attributes(5)
The `mlib_ImageSConv3x3()` function performs a separable 3x3 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1} \sum_{q=-dn}^{n-1} \text{src}[x+p][y+q][i]*h[p]*v[q]*2^{(-2*\text{scale})}
\]

where \(m = 3, n = 3, dm = (m - 1)/2 = 1, dn = (n - 1)/2 = 1\).

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `hkernel` Pointer to the horizontal kernel.
- `vkernel` Pointer to the vertical kernel.
- `scale` Scaling factor.
- `cmask` Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- `edge` Type of edge condition. It can be one of the following:

  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB),
mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB),
mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageSConv3x3_Fp()` function performs a separable 3x3 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-\text{dm}}^{\text{dm}} \sum_{q=-\text{dn}}^{\text{dn}} \text{src}[x+p][y+q][i] \times h[p] \times v[q]
\]

where \( m = 3 \), \( n = 3 \), \( \text{dm} = (m - 1)/2 = 1 \), \( \text{dn} = (n - 1)/2 = 1 \).

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **hkernel**: Pointer to the horizontal kernel.
- **vkernel**: Pointer to the vertical kernel.
- **cmask**: Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB),
mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB),
mlib_ImageConvolveMxN Fp(3MLIB), mlib_ImageSConv3x3(3MLIB),
mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
# Name

mlib_ImageSConv5x5 – separable 5x5 convolution

## Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSConv5x5(mlib_image *dst, const mlib_image *src,
                              const mlib_s32 *hkernel, const mlib_s32 *vkernel, mlib_s32 scale,
                              mlib_s32 cmask, mlib_edge edge);
```

## Description

The `mlib_ImageSConv5x5()` function performs a separable 5x5 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] \cdot h[p] \cdot v[q] \cdot 2^{(-2 \cdot \text{scale})}
\]

where \(m = 5\), \(n = 5\), \(dm = (m - 1)/2 = 2\), \(dn = (n - 1)/2 = 2\).

## Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **hkernel**: Pointer to the horizontal kernel.
- **vkernel**: Pointer to the vertical kernel.
- **scale**: Scaling factor.
- **cmask**: Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge**: Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_DST_COPY_SRC`
  - `MLIB_EDGE_SRC_EXTEND`

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
mlib_imageSConv5x5(3MLIB)

See Also  
mlib_imageConv2x2(3MLIB), mlib_imageConv2x2_Fp(3MLIB),
mlib_imageConv2x2Index(3MLIB), mlib_imageConv3x3(3MLIB),
mlib_imageConv3x3_Fp(3MLIB), mlib_imageConv3x3Index(3MLIB),
mlib_imageConv4x4(3MLIB), mlib_imageConv4x4_Fp(3MLIB),
mlib_imageConv4x4Index(3MLIB), mlib_imageConv5x5(3MLIB),
mlib_imageConv5x5_Fp(3MLIB), mlib_imageConv5x5Index(3MLIB),
mlib_imageConv7x7(3MLIB), mlib_imageConv7x7_Fp(3MLIB),
mlib_imageConv7x7Index(3MLIB), mlib_imageConvKernelConvert(3MLIB),
mlib_imageConvMxN(3MLIB), mlib_imageConvMxN_Fp(3MLIB),
mlib_imageConvMxNIndex(3MLIB), mlib_imageConvolveMxN(3MLIB),
mlib_imageConvolveMxN_Fp(3MLIB), mlib_imageSConv3x3(3MLIB),
mlib_imageSConv3x3_Fp(3MLIB), mlib_imageSConv5x5_Fp(3MLIB),
mlib_imageSConv7x7(3MLIB), mlib_imageSConv7x7_Fp(3MLIB),
mlib_imageSConvKernelConvert(3MLIB), attributes(5)
The `mlib_ImageSConv5x5_Fp()` function performs a separable 5x5 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

\[
    \text{dst}[x][y][i] = \sum_{p=-dm}^{m-1-dm} \sum_{q=-dn}^{n-1-dn} \text{src}[x+p][y+q][i] \times h[p] \times v[q]
\]

where \( m = 5 \), \( n = 5 \), \( dm = (m - 1)/2 = 2 \), \( dn = (n - 1)/2 = 2 \).

**Parameters**

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **hkernel**: Pointer to the horizontal kernel.
- **vkernel**: Pointer to the vertical kernel.
- **cmask**: Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to one bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also:
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB),
mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB),
mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB),
mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB),
mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
mlib_ImageSConv7x7 – separable 7x7 convolution

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_ImageSConv7x7(mlib_image *dst, const mlib_image *src,
    const mlib_s32 *hkernel, const mlib_s32 *vkernel, mlib_s32 scale,
    mlib_s32 cmask, mlib_edge edge);
```

Description

The mlib_ImageSConv7x7() function performs a separable 7x7 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

\[
\text{dst}[x][y][i] = \sum_{p=-dm}^{dm-1} \sum_{q=-dn}^{dn-1} \text{src}[x+p][y+q][i] \cdot h[p] \cdot v[q] \cdot 2^{-2 \cdot \text{scale}}
\]

where \( m = 7 \), \( n = 7 \), \( dm = (m - 1)/2 = 3 \), and \( dn = (n - 1)/2 = 3 \).

Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **hkernel**: Pointer to the horizontal kernel.
- **vkernel**: Pointer to the vertical kernel.
- **scale**: Scaling factor.
- **cmask**: Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge**: Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also

mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB),
mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB),
mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB),
mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB),
mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
# mlib_ImageSConv7x7_Fp

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSConv7x7_Fp(mlib_image *dst, const mlib_image *src,
   const mlib_d64 *hkernel, const mlib_d64 *vkernel, mlib_s32 cmask,
   mlib_edge edge);
```

**Description**

The `mlib_ImageSConv7x7_Fp()` function performs a separable 7x7 convolution on the source image by using the user-supplied kernel.

It uses the following equation:

\[
dst[x][y][i] = \sum_{p=-dm}^{m-1} \sum_{q=-dn}^{n-1} src[x+p][y+q][i] \times h[p] \times v[q]
\]

where 
\[
m = 7, n = 7, dm = (m - 1)/2 = 3, dn = (n - 1)/2 = 3.
\]

**Parameters**

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **hkernel** Pointer to the horizontal kernel.
- **vkernel** Pointer to the vertical kernel.
- **cmask** Channel mask to indicate the channels to be convolved, each bit of which represents a channel in the image. The channels corresponding to 1 bits are those to be processed. For a single-channel image, the channel mask is ignored.
- **edge** Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_DST_COPY_SRC
  - MLIB_EDGE_SRC_EXTEND

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_ImageConv2x2(3MLIB), mlib_ImageConv2x2_Fp(3MLIB),
mlib_ImageConv2x2Index(3MLIB), mlib_ImageConv3x3(3MLIB),
mlib_ImageConv3x3_Fp(3MLIB), mlib_ImageConv3x3Index(3MLIB),
mlib_ImageConv4x4(3MLIB), mlib_ImageConv4x4_Fp(3MLIB),
mlib_ImageConv4x4Index(3MLIB), mlib_ImageConv5x5(3MLIB),
mlib_ImageConv5x5_Fp(3MLIB), mlib_ImageConv5x5Index(3MLIB),
mlib_ImageConv7x7(3MLIB), mlib_ImageConv7x7_Fp(3MLIB),
mlib_ImageConv7x7Index(3MLIB), mlib_ImageConvKernelConvert(3MLIB),
mlib_ImageConvMxN(3MLIB), mlib_ImageConvMxN_Fp(3MLIB),
mlib_ImageConvMxNIndex(3MLIB), mlib_ImageConvolveMxN(3MLIB),
mlib_ImageConvolveMxN_Fp(3MLIB), mlib_ImageSConv3x3(3MLIB),
mlib_ImageSConv3x3_Fp(3MLIB), mlib_ImageSConv5x5(3MLIB),
mlib_ImageSConv5x5_Fp(3MLIB), mlib_ImageSConv7x7(3MLIB),
mlib_ImageSConvKernelConvert(3MLIB), attributes(5)
**mlib_ImageSConvKernelConvert**

**Name**
mlib_ImageSConvKernelConvert – kernel conversion for separable convolution

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSConvKernelConvert(mlib_s32 *ihkernel,
              mlib_s32 *ivkernel, mlib_s32 *iscale,
              const mlib_d64 *fhkernel,
              const mlib_d64 *fvkernel, mlib_s32 m, mlib_s32 n, mlib_type type);
```

**Description**
The `mlib_ImageSConvKernelConvert()` function converts a floating-point separable convolution kernel to an integer kernel with its scaling factor, which is suitable to be used in separable convolution functions.

**Parameters**
The function takes the following arguments:
- `ihkernel` Pointer to integer horizontal kernel.
- `ivkernel` Pointer to integer vertical kernel.
- `iscale` Scaling factor of the integer convolution kernel.
- `fhkernel` Pointer to floating-point horizontal kernel.
- `fvkernel` Pointer to floating-point vertical kernel.
- `m` Width of the convolution kernel. `m` must be an odd number larger than 1.
- `n` Height of the convolution kernel. `n` must be an odd number larger than 1.
- `type` The image type.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_ImageSConv3x3(3MLIB), mlib_ImageSConv3x3_Fp(3MLIB),
  mlib_ImageSConv5x5(3MLIB), mlib_ImageSConv5x5_Fp(3MLIB),
  mlib_ImageSConv7x7(3MLIB), mlib_ImageSConv7x7_Fp(3MLIB),
  mlib_ImageConvKernelConvert(3MLIB), attributes(5)`
mlib_ImageSetFormat(3MLIB)

Name
mlib_ImageSetFormat – set format

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSetFormat(mlib_image *img, mlib_format format);

Description
The mlib_ImageSetFormat() function sets a new value for the format field of a mlib_image structure.

The data type of the image can be MLIB_BIT, MLIB_BYTE, MLIB_SHORT, MLIB_USHORT, MLIB_INT, MLIB_FLOAT, or MLIB_DOUBLE.

Parameters
The function takes the following arguments:

img Pointer to a mediaLib image structure.
format Image pixel format. It can be one of the following:

MLIB_FORMAT_UNKNOWN
MLIB_FORMAT_INDEXED
MLIB_FORMAT_GRAYSCALE
MLIB_FORMAT_RGB
MLIB_FORMAT_BGR
MLIB_FORMAT_ARGB
MLIB_FORMAT_ABGR
MLIB_FORMAT_PACKED_ARGB
MLIB_FORMAT_PACKED_ABGR
MLIB_FORMAT_GRAYSCALE_ALPHA
MLIB_FORMAT_RGBA

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageGetFormat(3MLIB), mlib_ImageCreate(3MLIB),
mlib_ImageCreateStruct(3MLIB), mlib_ImageCreateSubimage(3MLIB), attributes(5)
The `mlib_ImageSetPaddings()` function sets new values for the `paddings` field of the `mlib_image` structure as follows:

```c
img->paddings[0] = left;
img->paddings[1] = top;
img->paddings[2] = right;
img->paddings[3] = bottom;
```

By default, an image structure creation function, such as `mlib_ImageCreate()`, `mlib_ImageCreateStruct()`, or `mlib_ImageCreateSubimage()`, sets the `paddings` field of the `mlib_image` structure as follows:

```c
img->paddings[0] = 0;
img->paddings[1] = 0;
img->paddings[2] = 0;
img->paddings[3] = 0;
```

Note that this function is needed only when the edge condition `MLIB_EDGE_SRC_PADDED` is used.

The `mlib_image->paddings` field denotes the amount of paddings on each side of an image, from which the real image border can be seen. When `MLIB_EDGE_SRC_PADDED` is specified as the edge condition, a geometric function uses the "real" source image border for clipping the destination image.

**Parameters**

The function takes the following arguments:

- `img`: Pointer to image data structure.
- `left`: Number of columns padded on the left side.
- `top`: Number of rows padded on the top.
- `right`: Number of columns padded on the right side.
- `bottom`: Number of rows padded at the bottom.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:
mlib_ImageSetPaddings(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageGetPaddings(3MLIB), mlib_ImageCreate(3MLIB),  
mlib_ImageCreateStruct(3MLIB), mlib_ImageCreateSubimage(3MLIB),  
mlib_ImageAffine(3MLIB), attributes(5)
**Name**  
mlib_ImageSetStruct – set image data structure

**Synopsis**  
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSetStruct(mlib_image *image, mlib_type type,
  mlib_s32 channels, mlib_s32 width, mlib_s32 height, mlib_s32 stride,
  const void *datbuf);
```

**Description**  
The `mlib_ImageSetStruct()` function sets a mediaLib image data structure using parameters supplied by the user.

The `mlib_ImageSetStruct()` function returns `MLIB_FAILURE` if the supplied parameters do not pass the following sanity checks:

- image should not be `NULL`
- type should be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`
- channels should be between 1 and 4
- width should be greater than 0
- height should be greater than 0
- stride should be no less than width * channels * (size of type in bytes)
- datbuf should not be `NULL`

Whenever `MLIB_FAILURE` is returned, the original image data structure is not changed.

If the data buffer in the image data structure is not `NULL`, it is the user's responsibility to free it if necessary.

**Parameters**  
The function takes the following arguments:

- **image**  
  Pointer to the image data structure.
- **type**  
  Image data type. It can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, `MLIB_INT`, `MLIB_FLOAT`, or `MLIB_DOUBLE`.
- **channels**  
  Number of channels in the image.
- **width**  
  Width of image in pixels.
- **height**  
  Height of image in pixels.
- **stride**  
  Stride of each row of the data space in bytes.
- **datbuf**  
  Pointer to the image data buffer.
Return Values  MLIB_SUCCESS is returned if the image data structure is set successfully. MLIB_FAILURE is returned when the image data structure cannot be set according to the parameters supplied.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageCreate(3MLIB), mlib_ImageCreateSubimage(3MLIB), mlib_ImageCreateStruct(3MLIB), mlib_ImageResetStruct(3MLIB), mlib_ImageDelete(3MLIB), mlib_ImageSetFormat(3MLIB), mlib_ImageSetPaddings(3MLIB), attributes(5)
mlib_ImageSetSubimageStruct

Name  mlib_ImageSetSubimageStruct — set sub-image data structure

Synopsis  
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSetSubimageStruct(mlib_image *subimg,
    const mlib_image *img, mlib_s32 x, mlib_s32 y,
    mlib_s32 w, mlib_s32 h);
```

Description  The mlib_ImageSetSubimageStruct() function sets a sub-image's data structure using parameters supplied by the user.

The mlib_ImageSetSubimageStruct() function returns MLIB_FAILURE if the supplied parameters do not pass the following sanity checks:

- subimg != NULL
- img != NULL
- 0 < w ≤ mlib_ImageGetWidth(img)
- 0 < h ≤ mlib_ImageGetHeight(img)
- 0 ≤ x ≤ (mlib_ImageGetWidth(img) - w)
- 0 ≤ y ≤ (mlib_ImageGetHeight(img) - h)

Whenever MLIB_FAILURE is returned, the original image data structure is not changed.

Parameters  The function takes the following arguments:

- `subimg`  Pointer to the sub-image data structure.
- `img`  Pointer to the source image data structure.
- `x`  X coordinate of the left border in the source image.
- `y`  Y coordinate of the top border in the source image.
- `w`  Width of the sub-image.
- `h`  Height of the sub-image.

Return Values  MLIB_SUCCESS is returned if the image data structure is set successfully. MLIB_FAILURE is returned when the image data structure cannot be set according to the parameters supplied.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_ImageCreate(3MLIB), mlib_ImageCreateSubimage(3MLIB),
mlib_ImageCreateStruct(3MLIB), mlib_ImageSetStruct(3MLIB),
mlib_ImageResetStruct(3MLIB), mlib_ImageResetSubimageStruct(3MLIB),
mlib_ImageDelete(3MLIB), mlib_ImageSetFormat(3MLIB),
mlib_ImageSetPaddings(3MLIB), attributes(5)
Name  mlib_ImageSobel, mlib_ImageSobel_Fp – Sobel filter

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSobel(mlib_image *dst, const mlib_image *src,
                         mlib_s32 cmask, mlib_edge edge);
mlib_status mlib_ImageSobel_Fp(mlib_image *dst, const mlib_image *src,
                         mlib_s32 cmask, mlib_edge edge);

Description  Each function is a special case of the gradient filter, which is an edge detector which computes
the magnitude of the image gradient vector in two orthogonal directions. In this case, the
gradient filter uses specific horizontal and vertical masks.

The Sobel filter is one of the special cases of gradient filter using the following horizontal and
vertical masks:

hmask = { -1.0, 0.0, 1.0,
          -2.0, 0.0, 2.0,
          -1.0, 0.0, 1.0 }

vmask = { -1.0, -2.0, -1.0,
          0.0, 0.0, 0.0,
          1.0, 2.0, 1.0 }

Parameters  Each function takes the following arguments:

dst  Pointer to destination image.
src  Pointer to source image.
cmask  Channel mask to indicate the channels to be processed, each bit of which represents
        a channel in the image. The channels corresponding to 1 bits are those to be
        processed. For a single channel image, the channel mask is ignored.
edge  Type of edge condition. It can be one of the following:

        MLIB_EDGE_DST_NO_WRITE
        MLIB_EDGE_DS_FILL_ZERO
        MLIB_EDGE_DST_COPY_SRC
        MLIB_EDGE_SR_EXTEND

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
mlib_ImageSobel(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageGradient3x3(3MLIB), mlib_ImageGradient3x3_Fp(3MLIB), attributes(5)
The `mlib_ImageSqr_Fp()` function computes the floating-point square of each pixel in the source image.

It uses the following equation:

\[ \text{dst}[x][y][i] = \text{src}[x][y][i] \times \text{src}[x][y][i] \]

### Parameters
- `dst` Pointer to destination image.
- `src` Pointer to source image.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageSqr_Fp_Inp(3MLIB)`, `attributes(5)`
mlib_ImageSqr_Fp_Inp – square, in place

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSqr_Fp_Inp(mlib_image *srcdst);
```

Description

The `mlib_ImageSqr_Fp_Inp()` function computes the floating-point square of each pixel in the source image.

It uses the following equation:

```
srcdst[x][y][i] = srcdst[x][y][i] * srcdst[x][y][i]
```

Parameters

The function takes the following arguments:

- `srcdst`: Pointer to source and destination image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_ImageSqr_Fp(3MLIB), attributes(5)`
mlib_ImageSqrShift – square with shifting

Synopsis

cc [ flag... ] file... -lm [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSqrShift(mlib_image *dst, const mlib_image *src, mlib_s32 shift);

Description

The mlib_ImageSqrShift() function computes the square of each pixel in the source image and scales the result by the shift factor.

It uses the following equation:

dst[x][y][i] = src[x][y][i] * src[x][y][i] * 2**(-shift)

Parameters

The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

shift Right shifting factor. 0 ≤ shift ≤ 31.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageSqrShift_Inp(3MLIB), attributes(5)
mlib_ImageSqrShift_Inp - square with shifting, in place

Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageSqrShift_Inp(mlib_image *srcdst, mlib_s32 shift);
```

Description

The `mlib_ImageSqrShift_Inp()` function computes the square of each pixel in the source image and scales the result by the shift factor, in place.

It uses the following equation:

```
srcdst[x][y][i] = srcdst[x][y][i] * srcdst[x][y][i] * 2**(shift)
```

Parameters

- **srcdst**: Pointer to source and destination image.
- **shift**: Right shifting factor. \(0 \leq \text{shift} \leq 31\).

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageSqrShift(3MLIB)`, `attributes(5)`
The `mlib_ImageStdDev()` function computes the standard deviation for each channel in the source image. It uses the following equation:

\[
1 \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img}[x][y][i] - \text{mean}[i])^2 \]^{0.5}
\]

where, in the case of \( \text{mean} == \text{NULL}, \)

\[
1 \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]
\]

The function takes the following arguments:

- `sdev` Pointer to standard deviation array, whose size is the number of channels in the source image. `sdev[i]` contains the standard deviation of channel `i`.
- `img` Pointer to input image.
- `mean` Pointer to pre-computed mean array for each channel. (If NULL, it will be computed.) `mean[i]` contains the mean of channel `i`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageMean(3MLIB), mlib_ImageMean_Fp(3MLIB), mlib_ImageStdDev_Fp(3MLIB), attributes(5)`
mlib_ImageStdDev_Fp – image standard deviation

**Synopsis**

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageStdDev_Fp(mlib_d64 *sdev, const mlib_image *img,
const mlib_d64 *mean);

**Description**

The `mlib_ImageStdDev_Fp()` function computes the standard deviation for each channel in the floating-point source image.

It uses the following equation:

\[
\text{sdev}[i] = \frac{1}{wh} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (\text{img}[x][y][i] - \text{mean}[i])^2 \times 0.5
\]

where, in the case of `mean == NULL`,

\[
\text{mean}[i] = \frac{1}{wh} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \text{img}[x][y][i]
\]

**Parameters**

The function takes the following arguments:

- **sdev** Pointer to standard deviation array, whose size is the number of channels in the source image. `sdev[i]` contains the standard deviation of channel `i`.
- **img** Pointer to input image.
- **mean** Pointer to pre-computed mean array for each channel. (If NULL, it will be computed.) `mean[i]` contains the mean of channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageMean(3MLIB), mlib_ImageMean_Fp(3MLIB), mlib_ImageStdDev(3MLIB), attributes(5)`
The `mlib_ImageSub1_Fp_Inp` function subtracts the second floating-point source image from the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
src1dst[x][y][i] = src1dst[x][y][i] - src2[x][y][i]
\]

The function takes the following arguments:

- `src1dst`: Pointer to first source and destination image.
- `src2`: Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See also `mlib_ImageSub(3MLIB), mlib_ImageSub_Fp(3MLIB), mlib_ImageSub1_Inp(3MLIB), mlib_ImageSub2_Fp_Inp(3MLIB), mlib_ImageSub2_Inp(3MLIB), attributes(5)`
The `mlib_ImageSub1_Inp()` function subtracts the second source image from the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src1dst[x][y][i] = src1dst[x][y][i] - src2[x][y][i]
```

The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageSub(3MLIB), mlib_ImageSub_Fp(3MLIB), mlib_ImageSub1_Fp_Inp(3MLIB), mlib_ImageSub2_Fp_Inp(3MLIB), mlib_ImageSub2_Inp(3MLIB), attributes(5)`
The `mlib_ImageSub2_Fp_Inp()` function subtracts the second floating-point source image from the first floating-point source image on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src2dst[x][y][i] = src1[x][y][i] - src2dst[x][y][i]
```

Parameters

- `src2dst` Pointer to second source and destination image.
- `src1` Pointer to first source image.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_ImageSub(3MLIB), mlib_ImageSub_Fp(3MLIB), mlib_ImageSub1_Fp_Inp(3MLIB), mlib_ImageSub1_Inp(3MLIB), mlib_ImageSub2_Inp(3MLIB), attributes(5)`
The `mlib_ImageSub2_Inp()` function subtracts the second source image from the first source image on a pixel-by-pixel basis, in place.

It uses the following equation:

```
src2dst[x][y][i] = src1[x][y][i] - src2dst[x][y][i]
```

The function takes the following arguments:

- `src2dst` Pointer to second source and destination image.
- `src1` Pointer to first source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageSub2_Inp(3MLIB)`
# mlib_ImageSub

The `mlib_ImageSub()` function subtracts the second source image from the first source image on a pixel-by-pixel basis.

\[
dst[x][y][i] = src1[x][y][i] - src2[x][y][i]
\]

## Parameters

- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_ImageSub_Fp(3MLIB)`, `mlib_ImageSub1_Fp_Inp(3MLIB)`, `mlib_ImageSub1_Inp(3MLIB)`
- `mlib_ImageSub2_Fp_Inp(3MLIB)`, `mlib_ImageSub2_Inp(3MLIB)`
- `attributes(5)`
# mlib ImageSub_Fp

The `mlib_ImageSub_Fp()` function subtracts the second floating-point source image from the first floating-point source image on a pixel-by-pixel basis.

It uses the following equation:

\[
\text{dst}[x][y][i] = \text{src1}[x][y][i] - \text{src2}[x][y][i]
\]

### Parameters

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src1**: Pointer to first source image.
- **src2**: Pointer to second source image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

See also: `mlib_ImageSub(3MLIB), mlib_ImageSub1_Fp_Inp(3MLIB), mlib_ImageSub1_Inp(3MLIB), mlib_ImageSub2_Fp_Inp(3MLIB), mlib_ImageSub2_Inp(3MLIB), attributes(5)`
mlib_ImageSubsampleAverage with a box filter

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSubsampleAverage(mlib_image *dst,
    const mlib_image *src, mlib_d64 xscale, mlib_d64 yscale);

mlib_status mlib_ImageSubsampleAverage_Fp(mlib_image *dst,
    const mlib_image *src, mlib_d64 xscale, mlib_d64 yscale);

Each function scales an image down with an adaptive box filter.

The subsampling algorithm performs the scaling operation by averaging all the pixel values from a block in the source image that correspond to the destination pixel.

The width and height of the source block for a destination pixel are computed as:

\[
\text{blockX} = \lfloor 1.0 / \text{xscale} \rfloor; \\
\text{blockY} = \lfloor 1.0 / \text{yscale} \rfloor;
\]

If we denote a pixel's location in an image by its column number and row number (both counted from 0), the destination pixel at \((i, j)\) is backward mapped to the source block whose upper-left corner pixel is at \((xValues[i], yValues[j])\), where

\[
\text{xValues[i]} = \lfloor i / \text{xscale} + 0.5 \rfloor; \\
\text{yValues[j]} = \lfloor j / \text{yscale} + 0.5 \rfloor;
\]

The width and height of the filled area in the destination are restricted by

\[
\text{dstW} = \lfloor \text{srcWidth} * \text{xscale} \rfloor; \\
\text{dstH} = \lfloor \text{srcHeight} * \text{yscale} \rfloor;
\]

where \text{srcWidth} and \text{srcHeight} are width and height of the source image.

Since the block size in source is defined from scale factors with roundup, some blocks (the rightmost and the bottommost blocks) may overrun the border of the source image by 1 pixel. In this case, such blocks are moved by 1 pixel to left/up direction in order to be inside of the source image.

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **xscale** X scale factor. \(0.0 < \text{xscale} \leq 1.0\).
- **yscale** Y scale factor. \(0.0 < \text{yscale} \leq 1.0\).
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_ImageSubsampleBinaryToGray(3MLIB), mlib_ImageFilteredSubsample(3MLIB), mlib_ImageZoomTranslate(3MLIB), mlib_ImageZoomTranslate_Fp(3MLIB), attributes(5)
mlib_ImageSubsampleBinaryToGray – subsamples a binary image and converts it to a grayscale image

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageSubsampleBinaryToGray(mlib_image *dst, const mlib_image *src, mlib_d64 xscale, mlib_d64 yscale, const mlib_u8 *lutGray);

Description The mlib_ImageSubsampleBinaryToGray() function subsamples a binary (MLIB_BIT) image and converts it to a grayscale (MLIB_BYTE) image.

The subsampling algorithm performs the scaling operation by accumulating all the bits in the source image that correspond to the destination pixel and, based on the x and y scaling factors, reserving consecutive indexes in the colormap for the maximum number of gray levels possible in the destination image. The destination image pixel values of this function are either gray levels or indexes (if lutGray==NULL).

For representing the source block of pixels that is used to determine destination pixel values, the index 0 represents a block with no 1’s (all 0’s), the index 1 represents a block with a single 1, and so on. If the scaling factors require a fractional block of source pixels to determine a destination pixel value, the block size is rounded up. For example, if a 2.2-by-2.2 block of source pixels would be required to determine destination pixel values, a 3-by-3 block is used, resulting in 10 possible gray levels and therefore 10 colormap indexes, whose values are 0 through 9.

The width and height of the source block for a destination pixel are computed as:

```
blockX = (int)ceil(1.0/xscale);
bblockY = (int)ceil(1.0/yscale);
```

If we denote a pixel’s location in an image by its column number and row number (both counted from 0), the destination pixel at (i, j) is backward mapped to the source block whose upper-left corner pixel is at (xValues[i], yValues[j]), where

```
xValues[i] = (int)(i/xscale + 0.5);
yValues[j] = (int)(j/yscale + 0.5);
```

The width and height of the filled area in the destination are restricted by

```
dstW = (int)(srcWidth * xscale);
dstH = (int)(srcHeight * yscale);
```

where srcWidth and srcHeight are width and height of the source image.

Since the block size in source is defined from scale factors with roundup, some blocks (the rightmost and the bottommost blocks) may overrun the border of the source image by 1 pixel. In this case, such blocks are moved by 1 pixel to left/up direction in order to be inside of the source image.
The function takes the following arguments:

**dst**  
Pointer to destination image. It must be of type `MLIB_BYTE` and have just one channel.

**src**  
Pointer to source image. It must be of type `MLIB_BIT` and have just one channel.

**xscale**  
X scale factor. 0.0 < xscale ≤ 1.0.

**yscale**  
Y scale factor. 0.0 < yscale ≤ 1.0.

**lutGray**  
Pointer to a grayscale lookup-table.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Parameters**

**Return Values**

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_ImageZoomTranslateToGray(3MLIB), mlib_ImageSubsampleAverage(3MLIB), attributes(5)
Name  mlib_ImageTestFlags – test flags

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

int mlib_ImageTestFlags(const mlib_image *img, mlib_s32 flags);

Description  The mlib_ImageTestFlags() function tests the flags for a combination of the following predefined characteristics. Note that the result of zero means the conditions are satisfied.

MLIB_IMAGE_ALIGNED64 /* data address is 64-byte aligned */
MLIB_IMAGE_ALIGNED8 /* data address is 8-byte aligned */
MLIB_IMAGE_ALIGNED4 /* data address is 4-byte aligned */
MLIB_IMAGE_ALIGNED2 /* data address is 2-byte aligned */
MLIB_IMAGE_WIDTH8X /* width is multiple of 8 */
MLIB_IMAGE_WIDTH4X /* width is multiple of 4 */
MLIB_IMAGE_WIDTH2X /* width is multiple of 2 */
MLIB_IMAGE_HEIGHT8X /* height is multiple of 8 */
MLIB_IMAGE_HEIGHT4X /* height is multiple of 4 */
MLIB_IMAGE_HEIGHT2X /* height is multiple of 2 */
MLIB_IMAGE_STRIDE8X /* stride is multiple of 8 */
MLIB_IMAGE_ONEVECTOR /* stride is equal to width in bytes */
MLIB_IMAGE_USERALLOCATED /* data space has been allocated by user */
MLIB_IMAGE_ATTRIBUTESET /* image attribute flags have been set */

Parameters  The function takes the following arguments:

  img  Pointer to a mediaLib image structure.

  flags  Combination of a set of characteristics to be tested. It is formed by logically Oring one or more individual predefined characteristics.

Return Values  The function returns an integer value containing results of test. Condition = 0 if satisfied; otherwise, Condition != 0.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageGetFlags(3MLIB), attributes(5)
mlib_ImageThresh1 – image thresholding

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageThresh1(mlib_image *dst, const mlib_image *src,
const mlib_s32 *thresh, const mlib_s32 *ghigh, const mlib_s32 *glow);

Description

The mlib_ImageThresh1() function compares each pixel in the source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

The data type of the destination image can be MLIB_BIT or can be the same as the data type of the source image.

It uses the following equation:

dst[x][y][i] = glow[i] if src[x][y][i] ≤ thresh[i]
dst[x][y][i] = ghigh[i] if src[x][y][i] > thresh[i]

Parameters

The function takes the following arguments:

dst Pointer to destination image.
src Pointer to source image.
thresh Threshold value. thresh[i] contains the threshold for channel i.
ghigh High output level. ghigh[i] contains the high output level for channel i.
glow Low output level. glow[i] contains the low output level for channel i.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB),
mlib_ImageThresh1(3MLIB)

mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
#include <mlib.h>

mlib_status mlib_ImageThresh1_Fp(mlib_image *dst, const mlib_image *src, 
const mlib_d64 *thresh, const mlib_d64 *ghigh, const mlib_d64 *glow);

The `mlib_ImageThresh1_Fp()` function compares each pixel in the floating-point source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

The data type of the destination image can be `MLIB_BIT` or can be the same as the data type of the source image.

It uses the following equation:

\[
\text{dst}[x][y][i] = \text{glow}[i] \quad \text{if} \quad \text{src}[x][y][i] \leq \text{thresh}[i] \\
\text{dst}[x][y][i] = \text{ghigh}[i] \quad \text{if} \quad \text{src}[x][y][i] > \text{thresh}[i]
\]

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `thresh` Threshold value. `thresh[i]` contains the threshold for channel `i`.
- `ghigh` High output level. `ghigh[i]` contains the high output level for channel `i`.
- `glow` Low output level. `glow[i]` contains the low output level for channel `i`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB)`
mlib_ImageThresh1_Fp

mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
Name mlib_ImageThresh1_Fp_Inp – image thresholding

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageThresh1_Fp_Inp(mlib_image *srcdst,
const mlib_d64 *thresh, const mlib_d64 *ghigh, const mlib_d64 *glow);

Description

The mlib_ImageThresh1_Fp_Inp() function compares each pixel in the floating-point source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

srcdst[x][y][i] = glow[i] if srcdst[x][y][i] ≤ thresh[i]
srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thresh[i]

Parameters

The function takes the following arguments:

srcdst Pointer to source and destination image.
thresh threshold value. thresh[i] contains the threshold for channel i.
ghigh High output level. ghigh[i] contains the high output level for channel i.
glow Low output level. glow[i] contains the low output level for channel i.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
<td></td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
<td></td>
</tr>
</tbody>
</table>

See Also

mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB),
mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB),
mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB),
mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB),
mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB),
mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB),
mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB),
mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
The `mlib_ImageThresh1_Inp()` function compares each pixel in the image to a threshold value on a per-channel basis. If the pixel is less than or equal to the threshold value, then it is reset to the low output level. If the pixel is greater than the threshold value, then it is reset to the high output level.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \text{glow}[i] \quad \text{if} \quad \text{srcdst}[x][y][i] \leq \text{thresh}[i] \\
\text{srcdst}[x][y][i] = \text{ghigh}[i] \quad \text{if} \quad \text{srcdst}[x][y][i] > \text{thresh}[i]
\]

**Parameters**

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `thresh` Threshold value. `thresh[i]` contains the threshold for channel `i`.
- `ghigh` High output level. `ghigh[i]` contains the high output level for channel `i`.
- `glow` Low output level. `glow[i]` contains the low output level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5)`
The `mlib_ImageThresh2` function compares each pixel in the source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
\text{dst}[x][y][i] = \text{glow}[i] \quad \text{if} \quad \text{src}[x][y][i] \leq \text{thresh}[i] \\
\text{dst}[x][y][i] = \text{src}[x][y][i] \quad \text{if} \quad \text{src}[x][y][i] > \text{thresh}[i]
\]

**Parameters**

The function takes the following arguments:

- `dst`  
  Pointer to destination image.

- `src`  
  Pointer to source image.

- `thresh`  
  Threshold value. `thresh[i]` contains the threshold for channel `i`.

- `glow`  
  Low output level. `glow[i]` contains the low output level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5)`
mlib_ImageThresh2_Fp(3MLIB)

Name  mlib_ImageThresh2_Fp – image thresholding

Synopsis  cc [ flag... ] file... -lmlib [ library... ]    
          #include <mlib.h>

          mlib_status mlib_ImageThresh2_Fp(mlib_image *dst, const mlib_image *src,    
          const mlib_d64 *thresh, const mlib_d64 *glow);

Description  The mlib_ImageThresh2_Fp() function compares each pixel in the floating-point source    
              image to a threshold value. If the pixel is less than or equal to the    
              threshold value, then the destination pixel is set to the low output    
              level. If the pixel is greater than the threshold value, then the    
              destination pixel is set to the value of the source pixel.

It uses the following equation:

dst[x][y][i] = glow[i] if src[x][y][i] ≤ thresh[i]

dst[x][y][i] = src[x][y][i] if src[x][y][i] > thresh[i]

Parameters  The function takes the following arguments:

dst Pointer to destination image.

src Pointer to source image.

thresh Threshold value. thresh[i] contains the threshold for channel i.

glow Low output level. glow[i] contains the low output level for channel i.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB),    
mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB),    
mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB),    
mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh3(3MLIB),    
mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB),    
mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB),    
mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB),    
mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),    
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),    
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
The `mlib_ImageThresh2_Fp_Inp()` function compares each pixel in the floating-point source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \text{glow}[i] \text{ if srcdst}[x][y][i] \leq \text{thresh}[i]
\]

**Parameters**

- `srcdst`  
  Pointer to source and destination image.
- `thresh`  
  Threshold value. `thresh[i]` contains the threshold for channel `i`.
- `glow`  
  Low output level. `glow[i]` contains the low output level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageThresh1(3MLIB)`, `mlib_ImageThresh1_Fp(3MLIB)`, `mlib_ImageThresh1_Inp(3MLIB)`
- `mlib_ImageThresh2(3MLIB)`, `mlib_ImageThresh2_Fp(3MLIB)`, `mlib_ImageThresh2_Inp(3MLIB)`
- `mlib_ImageThresh3(3MLIB)`, `mlib_ImageThresh3_Fp(3MLIB)`
- `mlib_ImageThresh3_Inp(3MLIB)`
- `mlib_ImageThresh4(3MLIB)`, `mlib_ImageThresh4_Fp(3MLIB)`
- `mlib_ImageThresh4_Inp(3MLIB)`
- `mlib_ImageThresh5(3MLIB)`, `mlib_ImageThresh5_Fp(3MLIB)`
- `mlib_ImageThresh5_Inp(3MLIB)`
The `mlib_ImageThresh2_Inp()` function compares each pixel in the source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the low output level. If the pixel is greater than the threshold value, then the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
srcdst[x][y][i] = \text{glow}[i] \text{ if } srcdst[x][y][i] \leq \text{thresh}[i]
\]

### Parameters

- **`srcdst`**  
  Pointer to source and destination image.

- **`thresh`**  
  Threshold value. `thresh[i]` contains the threshold for channel `i`.

- **`glow`**  
  Low output level. `glow[i]` contains the low output level for channel `i`.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5)`
The `mlib_ImageThresh3()` function compares each pixel in the source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

\[
\text{dst}[x][y][i] = \begin{cases} 
\text{src}[x][y][i] & \text{if src}[x][y][i] \leq \text{thresh}[i] \\
\text{ghigh}[i] & \text{if src}[x][y][i] > \text{thresh}[i]
\end{cases}
\]

### Parameters

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `thresh` Threshold value. `thresh[i]` contains the threshold for channel `i`.
- `ghigh` High output level. `ghigh[i]` contains the high output level for channel `i`.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5)`
#include <mlib.h>

mlib_status mlib_ImageThresh3_Fp(mlib_image *dst, const mlib_image *src,
                                 const mlib_d64 *thresh, const mlib_d64 *ghigh);

The mlib_ImageThresh3_Fp() function compares each pixel in the floating-point source image to a threshold value. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][i] &= \text{src}[x][y][i] \quad \text{if} \quad \text{src}[x][y][i] \leq \text{thresh}[i] \\
\text{dst}[x][y][i] &= \text{ghigh}[i] \quad \text{if} \quad \text{src}[x][y][i] > \text{thresh}[i]
\end{align*}
\]

**Parameters**

The function takes the following arguments:

- **dst**: Pointer to destination image.
- **src**: Pointer to source image.
- **thresh**: Threshold value. \(\text{thresh}[i]\) contains the threshold for channel \(i\).
- **ghigh**: High output level. \(\text{ghigh}[i]\) contains the high output level for channel \(i\).

**Return Values**

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5)
mlib_ImageThresh3_Fp_Inp(3MLIB)

**Name** mlib_ImageThresh3_Fp_Inp – image thresholding

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageThresh3_Fp_Inp(mlib_image * srcdst,
    const mlib_d64 * thresh, const mlib_d64 * ghigh);
```

**Description**
The `mlib_ImageThresh3_Fp_Inp()` function compares each pixel in the floating-point source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:
```
srcdst[x][y][i] = ghigh[i] if srcdst[x][y][i] > thresh[i]
```

**Parameters**
The function takes the following arguments:

- **srcdst** Pointer to source and destination image.
- **thresh** Threshold value. `thresh[i]` contains the threshold for channel `i`.
- **ghigh** High output level. `ghigh[i]` contains the high output level for channel `i`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB),
mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB),
mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB),
mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB),
mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB),
mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB),
mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB),
mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
The `mlib_ImageThresh3_Inp()` function compares each pixel in the source image to a threshold value, in place. If the pixel is less than or equal to the threshold value, then the destination pixel is set to the value of the source pixel. If the pixel is greater than the threshold value, then the destination pixel is set to the high output level.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \text{ghigh}[i] \text{ if } \text{srcdst}[x][y][i] > \text{thresh}[i]
\]

**Parameters**

The function takes the following arguments:

- `srcdst` Pointer to source and destination image.
- `thresh` Threshold value. `thresh[i]` contains the threshold for channel `i`.
- `ghigh` High output level. `ghigh[i]` contains the high output level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), attributes(5)`
# mlib_ImageThresh4(3MLIB)

## Name
mlib_ImageThresh4 – image thresholding

## Synopsis
```
c c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
```
```
mlib_status mlib_ImageThresh4(mlib_image *dst, const mlib_image *src,
                           const mlib_s32 *thigh, const mlib_s32 *tlow,
                           const mlib_s32 *ghigh,
                           const mlib_s32 *glow);
```

## Description
The `mlib_ImageThresh4()` function compares each pixel in the source image to two threshold values, `tlow` and `thigh`. If the pixel is less than the lower threshold value, `tlow`, then the destination pixel is set to the lower output level, `glow`. If the pixel is greater than the higher threshold value, `thigh`, then the destination pixel is set to the higher output level, `ghigh`. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:
```
dst[x][y][i] = glow[i]  if src[x][y][i] < tlow[i]
dst[x][y][i] = src[x][y][i]  if tlow[i] <= src[x][y][i] <= thigh[i]
dst[x][y][i] = ghigh[i]  if src[x][y][i] > thigh[i]
```

## Parameters
The function takes the following arguments:
- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **thigh** High threshold value. `thigh[i]` holds the high threshold for channel `i`.
- **tlow** Low threshold value. `tlow[i]` holds the low threshold for channel `i`.
- **ghigh** High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel `i`.
- **glow** Low output grayscale level. `glow[i]` holds the low output grayscale level for channel `i`.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageThresh1(3MLIB)`, `mlib_ImageThresh1_Fp(3MLIB)`
- `mlib_ImageThresh1_Fp_Inp(3MLIB)`, `mlib_ImageThresh1_Inp(3MLIB)`
- `mlib_ImageThresh2(3MLIB)`, `mlib_ImageThresh1_Inp(3MLIB)`
- `mlib_ImageThresh2_Fp(3MLIB)`
- `mlib_ImageThresh2_Fp_Inp(3MLIB)`
- `mlib_ImageThresh2_Inp(3MLIB)`

Multimedia Library Functions - Part 3

675
mlib_ImageThresh4(3MLIB)

mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB),
mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB),
mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB),
mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
# Name
mlib_ImageThresh4_Fp – image thresholding

## Synopsis
```c
#include <mlib.h>

mlib_status mlib_ImageThresh4_Fp(mlib_image *dst, const mlib_image *src,
                                const mlib_d64 *thigh, const mlib_d64 *tlow,
                                const mlib_d64 *ghigh,
                                const mlib_d64 *glow);
```

## Description
The `mlib_ImageThresh4_Fp()` function compares each pixel in the source image to two threshold values, `tlow` and `thigh`. If the pixel is less than the lower threshold value, `tlow`, then the destination pixel is set to the lower output level, `glow`. If the pixel is greater than the higher threshold value, `thigh`, then the destination pixel is set to the higher output level, `ghigh`. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

- \[ \text{dst}[x][y][i] = \text{glow}[i] \quad \text{if} \quad \text{src}[x][y][i] < \text{tlow}[i] \]
- \[ \text{dst}[x][y][i] = \text{src}[x][y][i] \quad \text{if} \quad \text{tlow}[i] \leq \text{src}[x][y][i] \leq \text{thigh}[i] \]
- \[ \text{dst}[x][y][i] = \text{ghigh}[i] \quad \text{if} \quad \text{src}[x][y][i] > \text{thigh}[i] \]

## Parameters
The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **thigh** High threshold value. `thigh[i]` holds the high threshold for channel `i`.
- **tlow** Low threshold value. `tlow[i]` holds the low threshold for channel `i`.
- **ghigh** High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel `i`.
- **glow** Low output grayscale level. `glow[i]` holds the low output grayscale level for channel `i`.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_ImageThresh1(3MLIB)`, `mlib_ImageThresh1_Fp(3MLIB)`,
- `mlib_ImageThresh1_Fp_Inp(3MLIB)`, `mlib_ImageThresh1_Inp(3MLIB)`
- `mlib_ImageThresh2(3MLIB)`, `mlib_ImageThresh1_Inp(3MLIB)`
- `mlib_ImageThresh2_Fp(3MLIB)`, `mlib_ImageThresh2_Fp_Inp(3MLIB)`,
- `mlib_ImageThresh2_Inp(3MLIB)`

Multimedia Library Functions - Part 3 677
mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB),
mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB),
mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB),
mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
The `mlib_ImageThresh4_Fp_Inp()` function compares each pixel in the source image to two threshold values, `tlow` and `thigh`. If the pixel is less than the lower threshold value, `tlow`, then the destination pixel is set to the lower output level, `glow`. If the pixel is greater than the higher threshold value, `thigh`, then the destination pixel is set to the higher output level, `ghigh`. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
\text{src}[x][y][i] = \text{glow}[i] \quad \text{if} \quad \text{src}[x][y][i] < \text{tlow}[i] \\
\text{src}[x][y][i] = \text{ghigh}[i] \quad \text{if} \quad \text{src}[x][y][i] > \text{thigh}[i]
\]

**Parameters**

- `srcdst`  
  Pointer to source and destination image.
- `thigh`  
  High threshold value. `thigh[i]` holds the high threshold for channel `i`.
- `tlow`  
  Low threshold value. `tlow[i]` holds the low threshold for channel `i`.
- `ghigh`  
  High output grayscale level. `ghigh[i]` holds the high output grayscale level for channel `i`.
- `glow`  
  Low output grayscale level. `glow[i]` holds the low output grayscale level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB),`
mlib_ImageThresh4_Fp_Inp(3MLIB)

mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB),
mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
#include <mlib.h>

mlib_status mlib_ImageThresh4_Inp(mlib_image *srcdst, 
const mlib_s32 *thigh, const mlib_s32 *tlow, const mlib_s32 *ghigh, 
const mlib_s32 *glow);

Description
The mlib_ImageThresh4_Inp() function compares each pixel in the source image to two threshold values, \textit{tlow} and \textit{thigh}. If the pixel is less than the lower threshold value, \textit{tlow}, then the destination pixel is set to the lower output level, \textit{glow}. If the pixel is greater than the higher threshold value, \textit{thigh}, then the destination pixel is set to the higher output level, \textit{ghigh}. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:
\[
\text{srcdst}[x][y][i] = \text{glow}[i] \quad \text{if} \quad \text{srcdst}[x][y][i] < \text{tlow}[i] \\
\text{srcdst}[x][y][i] = \text{ghigh}[i] \quad \text{if} \quad \text{srcdst}[x][y][i] > \text{thigh}[i]
\]

Parameters
The function takes the following arguments:
- \textit{srcdst} Pointer to source and destination image.
- \textit{thigh} High threshold value. \textit{thigh}[i] holds the high threshold for channel \textit{i}.
- \textit{tlow} Low threshold value. \textit{tlow}[i] holds the low threshold for channel \textit{i}.
- \textit{ghigh} High output grayscale level. \textit{ghigh}[i] holds the high output grayscale level for channel \textit{i}.
- \textit{glow} Low output grayscale level. \textit{glow}[i] holds the low output grayscale level for channel \textit{i}.

Return Values
The function returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

Attributes
See attributes\texttt{(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), 
mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), 
mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), 
mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), 
mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), 
mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB),
mlib_ImageThresh4_Inp(3MLIB)

mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB),
mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh5(3MLIB),
mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
mlib_ImageThresh5 — image thresholding

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageThresh5(mlib_image *dst, const mlib_image *src,
                               const mlib_s32 *thigh, const mlib_s32 *tlow, const mlib_s32 *gray);
```

**Description**

The `mlib_ImageThresh5()` function compares each pixel in the source image to two threshold values, `tlow` and `thigh`. If the pixel is in between the lower threshold value, `tlow`, and the higher threshold value, `thigh`, (inclusive on both sides), then the destination pixel is set to the value `gray`. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
\begin{align*}
\text{dst}[x][y][i] &= \text{src}[x][y][i] \quad \text{if src}[x][y][i] < \text{tlow}[i] \\
\text{dst}[x][y][i] &= \text{gray}[i] \quad \text{if tlow}[i] \leq \text{src}[x][y][i] \leq \text{thigh}[i] \\
\text{dst}[x][y][i] &= \text{src}[x][y][i] \quad \text{if src}[x][y][i] > \text{thigh}[i]
\end{align*}
\]

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `thigh` High threshold value. `thigh[i]` holds the high threshold for channel `i`.
- `tlow` Low threshold value. `tlow[i]` holds the low threshold for channel `i`.
- `gray` Output grayscale level. `gray[i]` holds the output grayscale level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5)`
Name  mlib_ImageThresh5_Fp – image thresholding

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageThresh5_Fp(mlib_image *dst, const mlib_image *src,
const mlib_d64 *thigh, const mlib_d64 *tlow, const mlib_d64 *gray);

Description  The mlib_ImageThresh5_Fp() function compares each pixel in the source image to two threshold values, tlow and thigh. If the pixel is in between the lower threshold value, tlow, and the higher threshold value, thigh, (inclusive on both sides), then the destination pixel is set to the value gray. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
dst[x][y][i] = \begin{cases} 
src[x][y][i] & \text{if } src[x][y][i] < tlow[i] \\
gray[i] & \text{if } tlow[i] \leq src[x][y][i] \leq thigh[i] \\
src[x][y][i] & \text{if } src[x][y][i] > thigh[i]
\end{cases}
\]

Parameters  The function takes the following arguments:

- \( dst \)  Pointer to destination image.
- \( src \)  Pointer to source image.
- \( thigh \)  High threshold value. \( thigh[i] \) holds the high threshold for channel \( i \).
- \( tlow \)  Low threshold value. \( tlow[i] \) holds the low threshold for channel \( i \).
- \( gray \)  Output grayscale level. \( gray[i] \) holds the output grayscale level for channel \( i \).

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB),
mlib_ImageThresh1_Fp_Inp(3MLIB), mlib_ImageThresh1_Inp(3MLIB),
mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB),
mlib_ImageThresh2_Fp_Inp(3MLIB), mlib_ImageThresh2_Inp(3MLIB),
mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB),
mlib_ImageThresh3_Fp_Inp(3MLIB), mlib_ImageThresh3_Inp(3MLIB),
mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB),
mlib_ImageThresh4_Fp_Inp(3MLIB), mlib_ImageThresh4_Inp(3MLIB),
mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp_Inp(3MLIB),
mlib_ImageThresh5_Inp(3MLIB), attributes(5)
mlib_ImageThresh5_Fp_Inp is an image thresholding function. It compares each pixel in the source image to two threshold values, `tlow` and `thigh`. If the pixel is in between the lower threshold value, `tlow`, and the higher threshold value, `thigh`, (inclusive on both sides), then the destination pixel is set to the value `gray`. Otherwise, the destination pixel is set to the value of the source pixel.

The function takes the following equation:

\[
\text{srcdst}[x][y][i] = \text{gray}[i] \text{ if } \text{tlow}[i] \leq \text{srcdst}[x][y][i] \leq \text{thigh}[i]
\]

### Parameters
- `srcdst` Pointer to source and destination image.
- `thigh` High threshold value. `thigh[i]` holds the high threshold for channel `i`.
- `tlow` Low threshold value. `tlow[i]` holds the low threshold for channel `i`.
- `gray` Output grayscale level. `gray[i]` holds the output grayscale level for channel `i`.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_ImageThresh1(3MLIB)`, `mlib_ImageThresh1_Fp(3MLIB)`, `mlib_ImageThresh1_Fp_Inp(3MLIB)`
- `mlib_ImageThresh2(3MLIB)`, `mlib_ImageThresh2_Fp(3MLIB)`, `mlib_ImageThresh2_Fp_Inp(3MLIB)`
- `mlib_ImageThresh3(3MLIB)`, `mlib_ImageThresh3_Fp(3MLIB)`, `mlib_ImageThresh3_Fp_Inp(3MLIB)`
- `mlib_ImageThresh4(3MLIB)`, `mlib_ImageThresh4_Fp(3MLIB)`, `mlib_ImageThresh4_Fp_Inp(3MLIB)`
- `mlib_ImageThresh5(3MLIB)`, `mlib_ImageThresh5_Fp(3MLIB)`, `mlib_ImageThresh5_Fp_Inp(3MLIB)`
- attributes(5)
**Synopsis**

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageThresh5_Inp(mlib_image *srcdst,
const mlib_s32 *thigh, const mlib_s32 *tlow, const mlib_s32 *gray);
```

**Description**

The `mlib_ImageThresh5_Inp()` function compares each pixel in the source image to two threshold values, `tlow` and `thigh`. If the pixel is in between the lower threshold value, `tlow`, and the higher threshold value, `thigh`, (inclusive on both sides), then the destination pixel is set to the value `gray`. Otherwise, the destination pixel is set to the value of the source pixel.

It uses the following equation:

\[
\text{srcdst}[x][y][i] = \text{gray}[i] \text{ if } \text{tlow}[i] \leq \text{srcdst}[x][y][i] \leq \text{thigh}[i]
\]

**Parameters**

The function takes the following arguments:

- **srcdst** Pointer to source and destination image.
- **thigh** High threshold value. `thigh[i]` holds the high threshold for channel `i`.
- **tlow** Low threshold value. `tlow[i]` holds the low threshold for channel `i`.
- **gray** Output grayscale level. `gray[i]` holds the output grayscale level for channel `i`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageThresh1(3MLIB), mlib_ImageThresh1_Fp(3MLIB), mlib_ImageThresh1_Inp(3MLIB), mlib_ImageThresh2(3MLIB), mlib_ImageThresh2_Fp(3MLIB), mlib_ImageThresh2_Inp(3MLIB), mlib_ImageThresh3(3MLIB), mlib_ImageThresh3_Fp(3MLIB), mlib_ImageThresh3_Inp(3MLIB), mlib_ImageThresh4(3MLIB), mlib_ImageThresh4_Fp(3MLIB), mlib_ImageThresh4_Inp(3MLIB), mlib_ImageThresh5(3MLIB), mlib_ImageThresh5_Fp(3MLIB), mlib_ImageThresh5_Inp(3MLIB), attributes(5) |
The `mlib_ImageXor()` function computes the exclusive Or of the first source image with the second source image on a pixel-by-pixel basis.

It uses the following equation:
\[
dst[x][y][i] = src1[x][y][i] \oplus src2[x][y][i]
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The function takes the following arguments:
- `dst` Pointer to destination image.
- `src1` Pointer to first source image.
- `src2` Pointer to second source image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageXor_Inp(3MLIB), attributes(5)`
The `mlib_ImageXor_Inp()` function computes the exclusive Or of the first source image with the second source image on a pixel-by-pixel basis, in place.

It uses the following equation:

\[
\text{src1dst}[x][y][i] = \text{src1dst}[x][y][i] \oplus \text{src2}[x][y][i]
\]

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

**Parameters**
The function takes the following arguments:

- `src1dst` Pointer to first source and destination image.
- `src2` Pointer to second source image.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_ImageXor(3MLIB), attributes(5)`
`mlib_ImageXProj()` function computes the sum of the pixels in each column of the source image. The image must be a single-channel image. It uses the following equation:

$$h-1\\sum_{y=0} xproj[x] = \sum_{y=0} \text{img}[x][y][0]$$

where $x = 0, 1, \ldots, w - 1$.

### Parameters
The function takes the following arguments:

- `xproj` Pointer to X-projection vector, where length is equal to the number of columns in the source image (in other words, the image width).
- `img` Pointer to an input image.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
`mlib_ImageXProj_Fp(3MLIB), mlib_ImageYProj(3MLIB), mlib_ImageYProj_Fp(3MLIB), attributes(5)`
The `mlib_ImageXProj_Fp()` function computes the sum of the pixels in each column of the floating-point source image.

It uses the following equation:

\[
\text{xproj}[x] = \sum_{y=0}^{h-1} \text{img}[x][y][0]
\]

where \( x = 0, 1, \ldots, w - 1 \).

The function takes the following arguments:

- `xproj` Pointer to X-projection vector, where length is equal to the number of columns in the source image (in other words, the image width).
- `img` Pointer to an input image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageXProj(3MLIB), mlib_ImageYProj(3MLIB), mlib_ImageYProj_Fp(3MLIB), attributes(5)`
The `mlib_ImageYProj()` function computes the sum of the pixels in each row of the source image.

The image must be a single-channel image.

It uses the following equation:

\[
yproj[y] = \sum_{x=0}^{w-1} \text{img}[x][y][0]
\]

where \( y = 0, 1, \ldots, h - 1 \).

The function takes the following arguments:

- `yproj` Pointer to Y-projection vector, where length is equal to the number of rows in the source image (in other words, the image height).
- `img` Pointer to an input image.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_ImageXProj(3MLIB), mlib_ImageXProj_Fp(3MLIB), mlib_ImageYProj_Fp(3MLIB), attributes(5)`
The `mlib_ImageYProj_Fp()` function computes the sum of the pixels in each row of the floating-point source image.

The image must be a single-channel image.

It uses the following equation:

\[ y_{proj}[y] = \sum_{x=0}^{w-1} img[x][y][0] \]

where \( y = 0, 1, \ldots, h - 1 \).

### Parameters

- `yproj` Pointer to Y-projection vector, where length is equal to the number of rows in the source image (in other words, the image height).
- `img` Pointer to an input image.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_ImageXProj(3MLIB)`, `mlib_ImageXProj_Fp(3MLIB)`, `mlib_ImageYProj(3MLIB)`, `attributes(5)`
# mlib_ImageZoom

## Synopsis
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_ImageZoom(mlib_image *dst, const mlib_image *src,
                          mlib_d64 zoomx, mlib_d64 zoomy, mlib_filter filter, mlib_edge edge);
```

## Description
The `mlib_ImageZoom()` function will enlarge or minify the source image by the X and Y zoom factors. It uses the interpolation method as described by the resampling filter.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The center of the upper-left corner pixel of an image is located at `(0.5, 0.5)`.

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

## Parameters
The function takes the following arguments:
- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `zoomx` X zoom factor. `zoomx > 0.0`.
- `zoomy` Y zoom factor. `zoomy > 0.0`.
- `filter` Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_EXTEND_INDEF`
  - `MLIB_EDGE_SRC_PADDED`

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:
mlib_ImageZoom(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageZoom_Fp(3MLIB), mlib_ImageZoomIn2X(3MLIB),
          mlib_ImageZoomIn2X_Fp(3MLIB), mlib_ImageZoomIn2XIndex(3MLIB),
          mlib_ImageZoomIndex(3MLIB), mlib_ImageZoomOut2X(3MLIB),
          mlib_ImageZoomOut2X_Fp(3MLIB), mlib_ImageZoomOut2XIndex(3MLIB), attributes(5)
### Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageZoomBlend(mlib_image *dst, const mlib_image *src,
    mlib_d64 zoomx, mlib_d64 zoomy, mlib_filter filter, mlib_edge edge,
    mlib_blend blend, mlib_s32 alpha, mlib_s32 cmask);
```

### Description

The `mlib_ImageZoomBlend()` function will enlarge or minify the source image by the X and Y zoom factors and blend it with the destination image.

This function is a special case of `mlib_ImageZoomTranslateBlend()` with the center of the source image being mapped to the center of the destination image.

The center of the upper-left corner pixel of an image is considered to be located at \((0.5, 0.5)\).

Both `src` and `dst` must be of type `MLIB_BYTE`. They can have either 3 or 4 channels.

The `src` image cannot have width or height larger than 32767.

### Parameters

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to first source image.
- `zoomx` X zoom factor. `zoomx > 0.0`.
- `zoomy` Y zoom factor. `zoomy > 0.0`.
- `filter` Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_EXTEND_INDEF`
  - `MLIB_EDGE_SRC_PADDED`
- `blend` Type of alpha blending. It can be one of the following:
  - `MLIB_BLEND_GTK_SRC`
  - `MLIB_BLEND_GTK_SRC_OVER`
  - `MLIB_BLEND_GTK_SRC_OVER2`
- `alpha` Overall alpha for blending.
cmask  Channel mask to indicate the alpha channel.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageZoomTranslateBlend(3MLIB), mlib_ImageZoomTranslateTableBlend(3MLIB), attributes(5)
#include <mlib.h>

mlib_status mlib_ImageZoom_Fp(mlib_image *dst, const mlib_image *src, mlib_d64 zoomx, mlib_d64 zoomy, mlib_filter filter, mlib_edge edge);

The `mlib_ImageZoom_Fp()` function will enlarge or minify the floating-point source image by the X and Y zoom factors. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

### Parameters

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **zoomx** X zoom factor. zoomx > 0.0.
- **zoomy** Y zoom factor. zoomy > 0.0.
- **filter** Type of resampling filter. It can be one of the following:
  - MLIB_NEAREST
  - MLIB_BILINEAR
  - MLIB_BICUBIC
  - MLIB_BICUBIC2

- **edge** Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_OP_NEAREST
  - MLIB_EDGE_SRC_EXTEND
  - MLIB_EDGE_SRC_EXTEND_INDEF
  - MLIB_EDGE_SRC_PADDED

### Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
mlib_ImageZoom_Fp(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_ImageZoom(3MLIB), mlib_ImageZoomIn2X(3MLIB), mlib_ImageZoomIn2X_Fp(3MLIB), mlib_ImageZoomIn2XIndex(3MLIB), mlib_ImageZoomIndex(3MLIB), mlib_ImageZoomOut2X(3MLIB), mlib_ImageZoomOut2X_Fp(3MLIB), mlib_ImageZoomOut2XIndex(3MLIB), attributes(5)
The `mlib_ImageZoomIn2X()` function enlarges the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `filter` Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_ImageZoom(3MLIB), mlib_ImageZoom_Fp(3MLIB), mlib_ImageZoomIn2X_Fp(3MLIB), mlib_ImageZoomIn2XIndex(3MLIB), mlib_ImageZoomIndex(3MLIB), mlib_ImageZoomOut2X(3MLIB), mlib_ImageZoomOut2X_Fp(3MLIB), mlib_ImageZoomOut2XIndex(3MLIB), attributes(5)`
The `mlib_ImageZoomIn2X_Fp()` function enlarges the floating-point source image by a factor of two. It uses the interpolation method as described by the resampling filter. The center of the upper-left corner pixel of an image is located at (0.5, 0.5). The width and height of the destination image can be different from those of the source image. The center of the source image is mapped onto the center of the destination image.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `filter` Type of resampling filter. It can be one of the following:
  
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`

- `edge` Type of edge condition. It can be one of the following:
  
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_ImageZoom(3MLIB)`, `mlib_ImageZoom_Fp(3MLIB)`, `mlib_ImageZoomIn2X(3MLIB)`, `mlib_ImageZoomIn2XIndex(3MLIB)`, `mlib_ImageZoomOut2X(3MLIB)`, `mlib_ImageZoomOut2X_Fp(3MLIB)`, `mlib_ImageZoomOut2XIndex(3MLIB)`, `attributes(5)`
The `mlib_ImageZoomIn2XIndex()` function enlarges the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `filter` Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`
- `colormap` Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:
See Also  

*mlib_ImageZoom(3MLIB)*, *mlib_ImageZoom_Fp(3MLIB)*, *mlib_ImageZoomIn2X(3MLIB)*,
*mlib_ImageZoomIn2X_Fp(3MLIB)*, *mlib_ImageZoomIndex(3MLIB)*,
*mlib_ImageZoomOut2X(3MLIB)*, *mlib_ImageZoomOut2X_Fp(3MLIB)*,
*mlib_ImageZoomOut2XIndex(3MLIB)*, *attributes(5)*
The `mlib_ImageZoomIndex()` function will enlarge or minify the source image by the X and Y zoom factors. It uses the interpolation method as described by the resampling filter.

The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.

The center of the upper-left corner pixel of an image is located at (0.5, 0.5).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

The function takes the following arguments:

- `dst` Pointer to destination image.
- `src` Pointer to source image.
- `zoomx` X zoom factor. `zoomx > 0.0`.
- `zoomy` Y zoom factor. `zoomy > 0.0`.
- `filter` Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`
- `colormap` Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageZoom(3MLIB), mlib_ImageZoom_Fp(3MLIB), mlib_ImageZoomIn2X(3MLIB), mlib_ImageZoomIn2X_Fp(3MLIB), mlib_ImageZoomIn2XIndex(3MLIB), mlib_ImageZoomOut2X(3MLIB), mlib_ImageZoomOut2X_Fp(3MLIB), mlib_ImageZoomOut2XIndex(3MLIB), attributes(5)
mlib_ImageZoomOut2X (3MLIB)

**Name**  
mlib_ImageZoomOut2X - 0.5X zoom

**Synopsis**  
cc [ flag... ] file... -lmllib [ library... ]  
#include <mlib.h>

    mlib_status mlib_ImageZoomOut2X(mlib_image *dst, const mlib_image *src,  
                              mlib_filter filter, mlib_edge edge);

**Description**  
The `mlib_ImageZoomOut2X()` function minifies the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

The width and height of the destination image can be different from those of the source image.

The center of the source image is mapped onto the center of the destination image.

**Parameters**  
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

- **filter**  
  Type of resampling filter. It can be one of the following:
  
  MLIB_NEAREST  
  MLIB_BILINEAR  
  MLIB_BICUBIC  
  MLIB_BICUBIC2

- **edge**  
  Type of edge condition. It can be one of the following:
  
  MLIB_EDGE_DST_NO_WRITE  
  MLIB_EDGE_DST_FILL_ZERO  
  MLIB_EDGE_OP_NEAREST  
  MLIB_EDGE_SRC_EXTEND  
  MLIB_EDGE_SRC_PADDED

**Return Values**  
The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageZoom(3MLIB), mlib_ImageZoom_Fp(3MLIB), mlib_ImageZoomIn2X(3MLIB),  
mlib_ImageZoomIn2X_Fp(3MLIB), mlib_ImageZoomIn2XIndex(3MLIB),  
mlib_ImageZoomIndex(3MLIB), mlib_ImageZoomOut2X_Fp(3MLIB),  
mlib_ImageZoomOut2XIndex(3MLIB), attributes(5)
The `mlib_ImageZoomOut2X_Fp()` function minimizes the floating-point source image by a factor of two. It uses the interpolation method as described by the resampling filter. The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

The width and height of the destination image can be different from those of the source image. The center of the source image is mapped onto the center of the destination image.

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **filter** Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_ImageZoomIn2X(3MLIB)`
- `mlib_ImageZoom_Fp(3MLIB)`
- `mlib_ImageZoomOut2X(3MLIB)`
- `mlib_ImageZoomIn2X_Fp(3MLIB)`
- `mlib_ImageZoomIn2XIndex(3MLIB)`
- `mlib_ImageZoomIndex(3MLIB)`
- `mlib_ImageZoomOut2XIndex(3MLIB)`, `attributes(5)`
# mlib_ImageZoomOut2XIndex

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_ImageZoomOut2XIndex(mlib_image *dst,
    const mlib_image *src, mlib_filter filter, mlib_edge edge,
    const void *colormap);
```

## Description

The `mlib_ImageZoomOut2XIndex()` function miniﬁes the source image by a factor of two. It uses the interpolation method as described by the resampling filter.

- The image data type must be `MLIB_BYTE` or `MLIB_SHORT`.
- The center of the upper-left corner pixel of an image is located at `(0.5, 0.5)`.
- The width and height of the destination image can be different from those of the source image.
- The center of the source image is mapped onto the center of the destination image.

## Parameters

The function takes the following arguments:

- `dst` - Pointer to destination image.
- `src` - Pointer to source image.
- `filter` - Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- `edge` - Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND` 
  - `MLIB_EDGE_SRC_PADDED`
- `colormap` - Internal data structure for inverse color mapping. This data structure is generated by the `mlib_ImageColorTrue2IndexInit()` function.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
See Also  

mland_ImageZoom(3MLIB),mland_ImageZoom_Fp(3MLIB),mland_ImageZoomIn2X(3MLIB),mland_ImageZoomIn2X_Fp(3MLIB),mland_ImageZoomIn2XIndex(3MLIB),mland_ImageZoomIndex(3MLIB),mland_ImageZoomOut2X(3MLIB),mland_ImageZoomOut2X_Fp(3MLIB),attributes(5)
# mlib_ImageZoomTranslate

The `mlib_ImageZoomTranslate()` function will enlarge or minify the source image by the X and Y zoom factors, with translation. It uses the interpolation method as described by the resampling filter.

It uses the following equation for coordinate mapping:

\[
xd = zoomx \times xs + tx \\
yd = zoomy \times ys + ty
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_BIT`, `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dst</code></td>
<td>Pointer to destination image.</td>
</tr>
<tr>
<td><code>src</code></td>
<td>Pointer to source image.</td>
</tr>
<tr>
<td><code>zoomx</code></td>
<td>X zoom factor. <code>zoomx &gt; 0</code>.</td>
</tr>
<tr>
<td><code>zoomy</code></td>
<td>Y zoom factor. <code>zoomy &gt; 0</code>.</td>
</tr>
<tr>
<td><code>tx</code></td>
<td>X translation.</td>
</tr>
<tr>
<td><code>ty</code></td>
<td>Y translation.</td>
</tr>
<tr>
<td><code>filter</code></td>
<td>Type of resampling filter. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td><code>MLIB_NEAREST</code></td>
</tr>
<tr>
<td></td>
<td><code>MLIB_BILINEAR</code></td>
</tr>
<tr>
<td></td>
<td><code>MLIB_BICUBIC</code></td>
</tr>
<tr>
<td></td>
<td><code>MLIB_BICUBIC2</code></td>
</tr>
<tr>
<td><code>edge</code></td>
<td>Type of edge condition. It can be one of the following:</td>
</tr>
</tbody>
</table>
MLIB_EDGE_DST_NO_WRITE
MLIB_EDGE_DST_FILL_ZERO
MLIB_EDGE_OP_NEAREST
MLIB_EDGE_SRC_EXTEND
MLIB_EDGE_SRC_EXTEND_INDEF
MLIB_EDGE_SRC_PADDED

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageZoomTranslate_Fp(3MLIB), mlib_ImageAffine(3MLIB), mlib_ImageAffine_Fp(3MLIB), attributes(5)
The `mlib_ImageZoomTranslateBlend()` function will enlarge or minify the source image by the X and Y zoom factors, with translation, and blend it with the destination image.

It uses the following equation for coordinate mapping:

\[
xd = \text{zoomx} \times xs + tx \\
yd = \text{zoomy} \times ys + ty
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

The alpha blending is closely combined with the interpolation to achieve better performance. Part of alpha blending has to be performed before or together with the interpolation if the source image has an alpha channel. In that case, the color components of each neighboring source pixel which participates in the interpolation (\(src_r\) and etc.) have to be pre-multiplied by the alpha component of the same source pixel (\(src_a\)). After the interpolation, the overall alpha (\(alpha\)), the interpolated source alpha (\(interp_a\)) and the destination pixel’s original alpha (\(dst_a\), if any) are used to blend the interpolated source pixel (with components \(interp_r\) and etc.) with the destination pixel (with components \(dst_r\) and etc.).

The MLIB_BLEND_GTK_SRC blending is similar to the SRC rule of the Porter-Duff rules for image compositing. It is defined by

\[
C_d = C_s \\
A_d = A_s
\]

in general, and by the following formula for this function:

```c
if (interp_a != 0.0) {
    if (dst_has_alpha) {
        dst_r = interp_r/interp_a;
        dst_g = interp_g/interp_a;
        dst_b = interp_b/interp_a;
        dst_a = interp_a;
    } else {
        dst_r = interp_r;
        dst_g = interp_g;
        dst_b = interp_b;
    }
}
```
The MLIB_BLEND_GTK_SRC_OVER or MLIB_BLEND_GTK_SRC_OVER2 blending is similar to the SRC_OVER rule of the Porter-Duff rules for image compositing. It is defined by

\[ Cd = Cs + Cd \cdot (1 - As) \]
\[ Ad = As + Ad \cdot (1 - As) \]

in general, and by the following formula for this function:

\[ w = alpha \cdot interp_a + (1 - alpha) \cdot interp_a \cdot dst_a; \]
\[ \text{if } (w \neq 0.0) \{ \]
\[ \text{if } (\text{MLIB_BLEND_GTK_SRC_OVER}) \{ \]
\[ dst_r = (alpha \cdot interp_r + \\
\[ (1 - alpha) \cdot interp_r) \cdot dst_a \cdot dst_r) / w; \]
\[ dst_g = (alpha \cdot interp_g + \\
\[ (1 - alpha) \cdot interp_g) \cdot dst_a \cdot dst_g) / w; \]
\[ dst_b = (alpha \cdot interp_b + \\
\[ (1 - alpha) \cdot interp_b) \cdot dst_a \cdot dst_b) / w; \]
\[ dst_a = w; \]
\[ } \text{else } \}
\[ dst_r = 0; \]
\[ dst_g = 0; \]
\[ dst_b = 0; \]
\[ dst_a = 0; \]
\[ } \]

where \( alpha, src_a, interp_a \) and \( dst_a \) are assumed to be in the range of \([0.0, 1.0]\).

For an image with 4 channels, the first or the fourth channel is considered the alpha channel if \( cmask \) equals 8 or 1, respectively. An image with 3 channels is considered to have no alpha channel, which is equivalent to having an alpha channel filled with all 1.0, or 0xff in case of MLIB_BYTE, if the general formulas for blending shown above are used.

Both \( src \) and \( dst \) must be of type MLIB_BYTE. They can have either 3 or 4 channels.

The \( src \) image cannot have width or height larger than 32767.

**Parameters**

The function takes the following arguments:

- \( dst \): Pointer to destination image.
- \( src \): Pointer to first source image.
- \( zoomx \): X zoom factor. \( zoomx > 0.0 \).
zoomy
  Y zoom factor. zoomy > 0.0.

tx
  X translation.

ty
  Y translation.

filter
  Type of resampling filter. It can be one of the following:
  
  MLIB_NEAREST
  MLIB_BILINEAR
  MLIB_BICUBIC
  MLIB_BICUBIC2

edge
  Type of edge condition. It can be one of the following:
  
  MLIB_EDGE_DST_NO_WRITE
  MLIB_EDGE_DST_FILL_ZERO
  MLIB_EDGE_OP_NEAREST
  MLIB_EDGE_SRC_EXTEND
  MLIB_EDGE_SRC_EXTEND_INDEF
  MLIB_EDGE_SRC_PADDED

blend
  Type of alpha blending. It can be one of the following:
  
  MLIB_BLEND_GTK_SRC
  MLIB_BLEND_GTK_SRC_OVER
  MLIB_BLEND_GTK_SRC_OVER2

alpha
  Overall alpha for blending.

cmask
  Channel mask to indicate the alpha channel.

Return Values
  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
<td></td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
<td></td>
</tr>
</tbody>
</table>

See Also
  mlib_ImageZoomBlend(3MLIB), mlib_ImageZoomTranslateTableBlend(3MLIB), attributes(5)
The `mlib_ImageZoomTranslate_Fp()` function will enlarge or minify the floating-point source image by the X and Y zoom factors, with translation. It uses the interpolation method as described by the resampling filter.

It uses the following equation for coordinate mapping:

\[ xd = zoomx \times xs + tx \]
\[ yd = zoomy \times ys + ty \]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

### Parameters

- **dst**  Pointer to destination image.
- **src**  Pointer to source image.
- **zoomx**  X zoom factor. \(zoomx > 0\).
- **zoomy**  Y zoom factor. \(zoomy > 0\).
- **tx**  X translation.
- **ty**  Y translation.
- **filter**  Type of resampling filter. It can be one of the following:
  - `MLIB_NEAREST`
  - `MLIB_BILINEAR`
  - `MLIB_BICUBIC`
  - `MLIB_BICUBIC2`
- **edge**  Type of edge condition. It can be one of the following:
MLIB EDGE_DST_NO_WRITE
MLIB EDGE_DST_FILL_ZERO
MLIB EDGE_OP_NEAREST
MLIB EDGE_SRC_EXTEND
MLIB EDGE_SRC_EXTEND_INDEF
MLIB EDGE_SRC_PADDED

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageZoomTranslate(3MLIB), mlib_ImageAffine(3MLIB), mlib_ImageAffine_Fp(3MLIB), attributes(5)
# mlib_ImageZoomTranslateTable

## Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_ImageZoomTranslateTable(mlib_image *dst,
const mlib_image *src, mlib_d64 zoomx, mlib_d64 zoomy,
mlib_d64 tx, mlib_d64 ty, const void *interp_table,
mlib_edge edge);
```

## Description

The `mlib_ImageZoomTranslateTable()` function will enlarge or minify the source image by the X and Y zoom factors, with translation. It uses a table, `interp_table`, to do interpolation.

It uses the following equation for coordinate mapping:

\[
xd = zoomx*xs + tx \\
yd = zoomy*ys + ty
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The data type of the images can be `MLIB_BYTE`, `MLIB_SHORT`, `MLIB_USHORT`, or `MLIB_INT`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

## Parameters

The function takes the following arguments:

- **dst** Pointer to destination image.
- **src** Pointer to source image.
- **zoomx** X zoom factor. `zoomx > 0`.
- **zoomy** Y zoom factor. `zoomy > 0`.
- **tx** X translation.
- **ty** Y translation.
- **interp_table** Pointer to an interpolation table. The table is created by the `mlib_ImageInterpTableCreate()` function.
- **edge** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_EXTEND_INDEF`
  - `MLIB_EDGE_SRC_PADDED`
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_ImageInterpTableCreate(3MLIB), mlib_ImageInterpTableDelete(3MLIB), mlib_ImageZoomTranslateTable_Fp(3MLIB), mlib_ImageZoomTranslate(3MLIB), mlib_ImageZoomTranslate_Fp(3MLIB), attributes(5)
**Name**  
mlib_ImageZoomTranslateTableBlend – image scaling using interpolation table, combined with alpha blending

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  
mlib_status mlib_ImageZoomTranslateTableBlend(mlib_image *dst,  
const mlib_image *src, mlib_d64 zoomx, mlib_d64 zoomy,  
mlib_d64 tx, mlib_d64 ty, const void *table, mlib_edge edge,  
mlib_blend blend, mlib_s32 cmask);

**Description**  
The `mlib_ImageZoomTranslateTableBlend()` function will enlarge or minify the source image by the X and Y zoom factors, with translation, and blend it with the destination image.

It uses the following equation for coordinate mapping:

\[
\begin{align*}
xd &= zoomx \times xs + tx \\
yd &= zoomy \times ys + ty
\end{align*}
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

It is assumed that the overall alpha for controlling the blending between the source image and the destination image has been pre-multiplied to the interpolation table for better performance.

The alpha blending is closely combined with the interpolation to achieve better performance. Part of alpha blending has to be performed before or together with the interpolation if the source image has an alpha channel. In that case, the color components of each neighboring source pixel which participates in the interpolation \((src_r\) and etc.) have to be pre-multiplied by the alpha component of the same source pixel \((src_a)\). After the interpolation, the interpolated alpha \((interp_a,\) which has been multiplied by the overall alpha because of the pre-multiplied interpolation table\) and the destination pixel’s original alpha \((dst_a,\) if any) are used to blend the interpolated source pixel \((\text{with components interp_r and etc.})\) with the destination pixel \((\text{with components dst_r and etc.})\).

The `MLIB_BLEND_GTK_SRC` blending is similar to the SRC rule of the Porter-Duff rules for image compositing. It is defined by

\[
\begin{align*}
C_d &= C_s \\
A_d &= A_s
\end{align*}
\]

in general, and by the following formula for this function:

```c
if (interp_a != 0.0) {
    if (dst_has_alpha) {
        dst_r = interp_r/interp_a;
        dst_g = interp_g/interp_a;
    }
```
The MLIB_BLEND_GTK_SRC_OVER or MLIB_BLEND_GTK_SRC_OVER2 blending is similar to the SRC_OVER rule of the Porter-Duff rules for image compositing. It is defined by

\[
Cd = Cs + Cd*(1 - As)
\]
\[
Ad = As + Ad*(1 - As)
\]

in general, and by the following formula for this function:

\[
w = \text{interp}_a + (1 - \text{interp}_a)\text{dst}_a;
\]

if \(w \neq 0.0\) {
  \[
dst_r = (\text{interp}_r + (1 - \text{interp}_a)\text{dst}_a\text{dst}_r)/w;
  \]
  \[
dst_g = (\text{interp}_g + (1 - \text{interp}_a)\text{dst}_a\text{dst}_g)/w;
  \]
  \[
dst_b = (\text{interp}_b + (1 - \text{interp}_a)\text{dst}_a\text{dst}_b)/w;
  \]
  \[
dst_a = w;
  \]
} else if (MLIB_BLEND_GTK_SRC_OVER) {
  \[
dst_r = 0;
  \]
  \[
dst_g = 0;
  \]
  \[
dst_b = 0;
  \]
  \[
dst_a = 0;
  \]
}

where src_a, interp_a and dst_a are assumed to be in the range of \([0.0, 1.0]\).

For an image with 4 channels, the first or the fourth channel is considered the alpha channel if cmask equals 8 or 1, respectively. An image with 3 channels is considered to have no alpha channel, which is equivalent to having an alpha channel filled with all 1.0, or 0xff in case of MLIB_BYTE, if the general formulas for blending shown above are used.

Both src and dst must be of type MLIB_BYTE. They can have either 3 or 4 channels.

The src image cannot have width or height larger than 32767.
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to first source image.

- **zoomx**  
  X zoom factor. \( \text{zoomx} > 0.0 \).

- **zoomy**  
  Y zoom factor. \( \text{zoomy} > 0.0 \).

- **tx**  
  X translation.

- **ty**  
  Y translation.

- **table**  
  Pointer to interpolation table structure.

- **edge**  
  Type of edge condition. It can be one of the following:
  - MLIB_EDGE_DST_NO_WRITE
  - MLIB_EDGE_DST_FILL_ZERO
  - MLIB_EDGE_OP_NEAREST
  - MLIB_EDGE_SRC_EXTEND
  - MLIB_EDGE_SRC_EXTEND_INDEF
  - MLIB_EDGE_SRC_PADDED

- **blend**  
  Type of alpha blending. It can be one of the following:
  - MLIB_BLEND_GTK_SRC
  - MLIB_BLEND_GTK_SRC_OVER
  - MLIB_BLEND_GTK_SRC_OVER2

- **cmask**  
  Channel mask to indicate the alpha channel.

**Return Values**  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_ImageZoomBlend(3MLIB), mlib_ImageZoomTranslateBlend(3MLIB), mlib_ImageInterpTableCreate(3MLIB), attributes(5)
**Name**
mlib_ImageZoomTranslateTable_Fp – zoom, with translation, with table-driven interpolation

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```
mlib_status mlib_ImageZoomTranslateTable_Fp(mlib_image *dst,
const mlib_image *src, mlib_d64 zoomx, mlib_d64 zoomy,
mlib_d64 tx, mlib_d64 ty, const void *interp_table,
mlib_edge edge);
```

**Description**
The `mlib_ImageZoomTranslateTable_Fp()` function will enlarge or minify the floating-point source image by the X and Y zoom factors, with translation. It uses a table, `interp_table`, to do interpolation.

It uses the following equation for coordinate mapping:

- \( x_d = zoomx \times x_s + tx \)
- \( y_d = zoomy \times y_s + ty \)

where a point with coordinates \((x_s, y_s)\) in the source image is mapped to a point with coordinates \((x_d, y_d)\) in the destination image.

The data type of the images can be `MLIB_FLOAT` or `MLIB_DOUBLE`.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

**Parameters**
The function takes the following arguments:

- **dst**  
  Pointer to destination image.

- **src**  
  Pointer to source image.

- **zoomx**  
  X zoom factor. `zoomx > 0`.

- **zoomy**  
  Y zoom factor. `zoomy > 0`.

- **tx**  
  X translation.

- **ty**  
  Y translation.

- **interp_table**  
  Pointer to an interpolation table. The table is created by the `mlib_ImageInterpTableCreate()` function.

- **edge**  
  Type of edge condition. It can be one of the following:

  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_NEAREST`
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_ImageInterpTableCreate(3MLIB), mlib_ImageInterpTableDelete(3MLIB),
mlib_ImageZoomTranslateTable(3MLIB), mlib_ImageZoomTranslate(3MLIB),
mlib_ImageZoomTranslate_Fp(3MLIB), attributes(5)
The `mlib_ImageZoomTranslateToGray()` function will enlarge or minify the source binary image by the X and Y zoom factors, with translation, and convert the resulting image into a grayscale image.

It uses the following equation for coordinate mapping:

\[
xd = \text{zoomx} \times xs + tx \\
yd = \text{zoomy} \times ys + ty
\]

where a point with coordinates \((xs, ys)\) in the source image is mapped to a point with coordinates \((xd, yd)\) in the destination image.

The width and height of the destination image can be different from the width and height of the source image.

The center of the upper-left corner pixel of an image is located at \((0.5, 0.5)\).

The function takes the following arguments:

- **`dst`** Pointer to destination image. It must be of type `MLIB_BYTE` and have just one channel.
- **`src`** Pointer to source image. It must be of type `MLIB_BIT` and have just one channel.
- **`zoomx`** X zoom factor. `zoomx > 0`.
- **`zoomy`** Y zoom factor. `zoomy > 0`.
- **`tx`** X translation.
- **`ty`** Y translation.
- **`filter`** Type of resampling filter. It must be `MLIB_NEAREST`.
- **`edge`** Type of edge condition. It can be one of the following:
  - `MLIB_EDGE_DST_NO_WRITE`
  - `MLIB_EDGE_DST_FILL_ZERO`
  - `MLIB_EDGE_OP_NEAREST`
  - `MLIB_EDGE_SRC_EXTEND`
  - `MLIB_EDGE_SRC_PADDED`
- **`ghigh`** Pointer to value for ‘1’ pixels in source image.
glow    Pointer to value for '0' pixels in source image.

**Return Values**  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_ImageSubsampleBinaryToGray(3MLIB), attributes(5)
mlib malloc – allocate a block of bytes

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void *mlib_malloc(size_t size);

Description
The mlib_malloc() function allocates size bytes on a 16-byte aligned boundary and returns a pointer to the allocated block.

This function is equivalent to memalign(16, size).

Parameters
The function takes the following arguments:
size Size of the block in bytes.

Return Values
The function returns a pointer to the allocated block if successful. Otherwise it returns a null pointer.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_free(3MLIB), mlib_realloc(3MLIB), malloc(3C), attributes(5)
Name  mlib_MatrixAddS_U8_Mod, mlib_MatrixAddS_U8_Sat, mlib_MatrixAddS_U8C_Mod, mlib_MatrixAddS_U8C_Sat, mlib_MatrixAddS_S8_Mod, mlib_MatrixAddS_S8_Sat, mlib_MatrixAddS_S8C_Mod, mlib_MatrixAddS_S8C_Sat, mlib_MatrixAddS_S16_Mod, mlib_MatrixAddS_S16_Sat, mlib_MatrixAddS_S16C_Mod, mlib_MatrixAddS_S16C_Sat, mlib_MatrixAddS_S32_Mod, mlib_MatrixAddS_S32_Sat, mlib_MatrixAddS_S32C_Mod, mlib_MatrixAddS_S32C_Sat – matrix addition to scalar, in place

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixAddS_U8_Mod(mlib_u8 *xz, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_U8_Sat(mlib_u8 *xz, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_U8C_Mod(mlib_u8 *xz, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_U8C_Sat(mlib_u8 *xz, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8_Mod(mlib_s8 *xz, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8_Sat(mlib_s8 *xz, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8C_Mod(mlib_s8 *xz, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8C_Sat(mlib_s8 *xz, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_Mod(mlib_s16 *xz, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_Sat(mlib_s16 *xz, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_Mod(mlib_s16 *xz, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_Sat(mlib_s16 *xz, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_Mod(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_Sat(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_Mod(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c,
        mlib_s32 m, mlib_s32 n);

**Description**  Each of these functions performs an in-place addition of a scalar value to a matrix.

For real data, the following equation is used:
\[ xz[i] = c[0] + xz[i] \]
where \( i = 0, 1, \ldots, (m \times n - 1) \).

For complex data, the following equation is used:
\[
\begin{align*}
\end{align*}
\]
where \( i = 0, 1, \ldots, (m \times n - 1) \).

**Parameters**  Each of the functions takes the following arguments:
- \( xz \)  Pointer to the source and the destination matrix.
- \( c \)  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( m \)  Number of rows in the matrices.
- \( n \)  Number of columns in the matrices.

**Return Values**  Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_MatrixAddS_U8_U8_Mod(3MLIB)`, attributes(5)
**Name**
mlib_MatrixAddS_U8_U8_Mod, mlib_MatrixAddS_U8_U8_Sat,
mlib_MatrixAddS_U8C_U8C_Mod, mlib_MatrixAddS_U8C_U8C_Sat,
mlib_MatrixAddS_S8_S8_Mod, mlib_MatrixAddS_S8_S8_Sat,
mlib_MatrixAddS_S8C_S8C_Mod, mlib_MatrixAddS_S8C_S8C_Sat,
mlib_MatrixAddS_S16_U8_Mod, mlib_MatrixAddS_S16_U8_Sat,
mlib_MatrixAddS_S16_S8_Mod, mlib_MatrixAddS_S16_S8_Sat,
mlib_MatrixAddS_S16_S16_Mod, mlib_MatrixAddS_S16_S16_Sat,
mlib_MatrixAddS_S16C_U8C_Mod, mlib_MatrixAddS_S16C_U8C_Sat,
mlib_MatrixAddS_S16C_S8C_Mod, mlib_MatrixAddS_S16C_S8C_Sat,
mlib_MatrixAddS_S16C_S16C_Mod, mlib_MatrixAddS_S16C_S16C_Sat,
mlib_MatrixAddS_S32_S16_Mod, mlib_MatrixAddS_S32_S16_Sat,
mlib_MatrixAddS_S32_S32_Mod, mlib_MatrixAddS_S32_S32_Sat,
mlib_MatrixAddS_S32C_S16C_Mod, mlib_MatrixAddS_S32C_S16C_Sat,
mlib_MatrixAddS_S32C_S32C_Mod, mlib_MatrixAddS_S32C_S32C_Sat – matrix addition
to scalar

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixAddS_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8c *x,
    const mlib_u8c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8c *x,
    const mlib_u8c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8c *x,
    const mlib_s8c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8c *x,
    const mlib_s8c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16c *x,
    const mlib_s16c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16c *x,
    const mlib_s16c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
    const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
    const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16c *x,
    const mlib_s16c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16c *x,
    const mlib_s16c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32c *x,
    const mlib_s32c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32c *x,
    const mlib_s32c *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAddS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

Description
Each of these functions adds a scalar value to a matrix.

For real data, the following equation is used:

\[ z[i] = c[0] + x[i] \]
where \( i = 0, 1, \ldots, (m*n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
z[2*i] &= c[0] + x[2*i] \\
\end{align*}
\]

where \( i = 0, 1, \ldots, (m*n - 1) \).

**Parameters**  Each of the functions takes the following arguments:

- \( z \)  Pointer to the destination matrix.
- \( x \)  Pointer to the source matrix.
- \( c \)  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( m \)  Number of rows in the matrices.
- \( n \)  Number of columns in the matrices.

**Return Values**  Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_MatrixAddS_U8_U8_Mod(3MLIB)`, attributes(5)
**Name**
mlib_MatrixAdd_U8_Mod, mlib_MatrixAdd_U8_Sat, mlib_MatrixAdd_U8C_Mod, mlib_MatrixAdd_U8C_Sat, mlib_MatrixAdd_S8_Mod, mlib_MatrixAdd_S8_Sat, mlib_MatrixAdd_S8C_Mod, mlib_MatrixAdd_S8C_Sat, mlib_MatrixAdd_S16_Mod, mlib_MatrixAdd_S16_Sat, mlib_MatrixAdd_S16C_Mod, mlib_MatrixAdd_S16C_Sat, mlib_MatrixAdd_S32_Mod, mlib_MatrixAdd_S32_Sat, mlib_MatrixAdd_S32C_Mod, mlib_MatrixAdd_S32C_Sat – matrix addition, in place

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```
mlib_status mlib_MatrixAdd_U8_Mod(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_U8_Sat(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_U8C_Mod(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_U8C_Sat(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S8_Mod(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S8_Sat(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S8C_Mod(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S8C_Sat(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16_Mod(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16_Sat(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16C_Mod(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16C_Sat(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32_Mod(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32_Sat(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32C_Mod(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 m, mlib_s32 n);
```
```c
mlib_status mlib_MatrixAdd_S32C_Sat(mlib_s32 *xz,
        const mlib_s32 *y, mlib_s32 m, mlib_s32 n);
```

**Description**  
Each of these functions performs an in-place addition of the second source matrix to the first source matrix.  

It uses the following equation:  
\[ xz[i] = xz[i] + y[i] \]

where \( i = 0, 1, \ldots, (m*n - 1) \) for real data; \( i = 0, 1, \ldots, (m*n^2 - 1) \) for complex data.

**Parameters**  
Each of the functions takes the following arguments:  
- `xz` Pointer to the first source and destination matrix.  
- `y` Pointer to the second source matrix.  
- `m` Number of rows in the matrices.  
- `n` Number of columns in the matrices.

**Return Values**  
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_MatrixAdd_U8_U8_Mod(3MLIB), attributes(5)
mlib_MatrixAdd_U8_U8_Mod, mlib_MatrixAdd_U8_U8_Sat,
mlib_MatrixAdd_U8C_U8C_Mod, mlib_MatrixAdd_U8C_U8C_Sat,
mlib_MatrixAdd_S8_S8_Mod, mlib_MatrixAdd_S8_S8_Sat,
mlib_MatrixAdd_S8C_S8C_Mod, mlib_MatrixAdd_S8C_S8C_Sat,
mlib_MatrixAdd_S16_U8_Mod, mlib_MatrixAdd_S16_U8_Sat,
mlib_MatrixAdd_S16_S8_Mod, mlib_MatrixAdd_S16_S8_Sat,
mlib_MatrixAdd_S16C_S8C_Mod, mlib_MatrixAdd_S16C_S8C_Sat,
mlib_MatrixAdd_S32_S16_Mod, mlib_MatrixAdd_S32_S16_Sat,
mlib_MatrixAdd_S32_S32_Mod, mlib_MatrixAdd_S32_S32_Sat,
mlib_MatrixAdd_S32C_S32C_Mod, mlib_MatrixAdd_S32C_S32C_Sat

Name
mlib_MatrixAdd_U8_U8_Mod, mlib_MatrixAdd_U8_U8_Sat,
mlib_MatrixAdd_U8C_U8C_Mod, mlib_MatrixAdd_U8C_U8C_Sat,
mlib_MatrixAdd_S8_S8_Mod, mlib_MatrixAdd_S8_S8_Sat,
mlib_MatrixAdd_S8C_S8C_Mod, mlib_MatrixAdd_S8C_S8C_Sat,
mlib_MatrixAdd_S16_U8_Mod, mlib_MatrixAdd_S16_U8_Sat,
mlib_MatrixAdd_S16_S8_Mod, mlib_MatrixAdd_S16_S8_Sat,
mlib_MatrixAdd_S16C_S8C_Mod, mlib_MatrixAdd_S16C_S8C_Sat,
mlib_MatrixAdd_S32_S16_Mod, mlib_MatrixAdd_S32_S16_Sat,
mlib_MatrixAdd_S32_S32_Mod, mlib_MatrixAdd_S32_S32_Sat,
mlib_MatrixAdd_S32C_S32C_Mod, mlib_MatrixAdd_S32C_S32C_Sat

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixAdd_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
count mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
count mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
count mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
count mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
count mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
count mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
count mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
count mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
count mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x,
count mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
count mlib_s16 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
count mlib_s16 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
count mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
count mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x,
count mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixAdd_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x,
count mlib_s32 *y, mlib_s32 m, mlib_s32 n);
Each of these functions adds the first source matrix to the second source matrix and writes the output to the destination matrix.

It uses the following equation:

\[ \text{z[i]} = \text{x[i]} + \text{y[i]} \]
where \( i = 0, 1, \ldots, (m\times n - 1) \) for real data; \( i = 0, 1, \ldots, (m\times n\times 2 - 1) \) for complex data.

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the destination matrix.
- \( x \) Pointer to the first source matrix.
- \( y \) Pointer to the second source matrix.
- \( m \) Number of rows in the matrices.
- \( n \) Number of columns in the matrices.

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_MatrixAdd_U8_U8_Mod(3MLIB), attributes(5)
Each of these functions performs an in-place averaging of two matrices. It uses the following equation:

\[ xz[i] = (xz[i] + y[i] + 1) / 2 \]

where \( i = 0, 1, \ldots, (m*n - 1) \) for real data; \( i = 0, 1, \ldots, (m*n^2 - 1) \) for complex data.

Each of the functions takes the following arguments:

- \( xz \) Pointer to the first source and destination matrix.
- \( y \) Pointer to the second source matrix.
- \( m \) Number of rows in the matrices.
- \( n \) Number of columns in the matrices.

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_MatrixAve_U8_U8(3MLIB), attributes(5)
Each of these functions computes the average of the first source matrix and the second source matrix and writes the output to the destination matrix.

It uses the following equation:
\[ z[i] = \frac{(x[i] + y[i] + 1)}{2} \]

where \( i = 0, 1, \ldots, (m*n - 1) \) for real data; \( i = 0, 1, \ldots, (m*n*2 - 1) \) for complex data.

**Parameters**  
Each of the functions takes the following arguments:

- \( z \)  
  Pointer to the destination matrix.
- \( x \)  
  Pointer to the first source matrix.
- \( y \)  
  Pointer to the second source matrix.
- \( m \)  
  Number of rows in the matrices.
- \( n \)  
  Number of columns in the matrices.

**Return Values**  
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_MatrixAve_U8_U8(3MLIB), attributes(5)`
Name  
mlib_MatrixMaximumMag_U8C, mlib_MatrixMaximumMag_S8C,  
mlib_MatrixMaximumMag_S16C, mlib_MatrixMaximumMag_S32C,  
mlib_MatrixMaximumMag_F32C, mlib_MatrixMaximumMag_D64C – find the first element  
with the maximum magnitude in a matrix

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

mlib_status mlib_MatrixMaximumMag_U8C(mlib_u8 *max, const mlib_u8 *x,  
mlib_s32 m, mlib_s32 n);  
mlib_status mlib_MatrixMaximumMag_S8C(mlib_s8 *max, const mlib_s8 *x,  
mlib_s32 m, mlib_s32 n);  
mlib_status mlib_MatrixMaximumMag_S16C(mlib_s16 *max, const mlib_s16 *x,  
mlib_s32 m, mlib_s32 n);  
mlib_status mlib_MatrixMaximumMag_S32C(mlib_s32 *max, const mlib_s32 *x,  
mlib_s32 m, mlib_s32 n);  
mlib_status mlib_MatrixMaximumMag_F32C(mlib_f32 *max, const mlib_f32 *x,  
mlib_s32 m, mlib_s32 n);  
mlib_status mlib_MatrixMaximumMag_D64C(mlib_d64 *max, const mlib_d64 *x,  
mlib_s32 m, mlib_s32 n);

Description  
Each of these functions finds the first element with the maximum magnitude in a complex  
matrix, then puts the real and imaginary parts of it into max[0] and max[1], respectively.

Parameters  
Each of the functions takes the following arguments:

max     Pointer to the first element with the maximum magnitude
x       Pointer to the first element of the source matrix.
m       Number of rows in the source matrix
n       Number of columns in the source matrix

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_MatrixMinimumMag_U8C(3MLIB), mlib_VectorMaximumMag_U8C(3MLIB),  
mlib_VectorMinimumMag_U8C(3MLIB), attributes(5)
find the maximum value in a matrix

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMaximum_U8(mlib_u8 *max, const mlib_u8 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMaximum_S8(mlib_s8 *max, const mlib_s8 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMaximum_S16(mlib_s16 *max, const mlib_s16 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMaximum_S32(mlib_s32 *max, const mlib_s32 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMaximum_F32(mlib_f32 *max, const mlib_f32 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMaximum_D64(mlib_d64 *max, const mlib_d64 *x, mlib_s32 m, mlib_s32 n);

Description

Each of these functions finds the maximum value of all elements in a matrix.

It uses the following equation:

\[ \text{max}[0] = \text{MAX} \{ x[i] \mid i = 0, 1, \ldots, (m*n - 1) \} \]

Parameters

Each of the functions takes the following arguments:

\[ \text{max} \quad \text{Pointer to the maximum value.} \]
\[ x \quad \text{Pointer to the first element of the source matrix.} \]
\[ m \quad \text{Number of rows in the source matrix.} \]
\[ n \quad \text{Number of columns in the source matrix.} \]

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

mlib_MatrixMaximum_U8(3MLIB), mlib_VectorMaximum_U8(3MLIB),
mlib_VectorMinimum_U8(3MLIB), attributes(5)
**Name**  
mlib_MatrixMinimumMag_U8C, mlib_MatrixMinimumMag_S8C, 
mlib_MatrixMinimumMag_S16C, mlib_MatrixMinimumMag_S32C, 
mlib_MatrixMinimumMag_F32C, mlib_MatrixMinimumMag_D64C – find the first element 
with the minimum magnitude in a matrix

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```
mlib_status mlib_MatrixMinimumMag_U8C(mlib_u8 *min, const mlib_u8 *x, 
  mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimumMag_S8C(mlib_s8 *min, const mlib_s8 *x, 
  mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimumMag_S16C(mlib_s16 *min, const mlib_s16 *x, 
  mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimumMag_S32C(mlib_s32 *min, const mlib_s32 *x, 
  mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimumMag_F32C(mlib_f32 *min, const mlib_f32 *x, 
  mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimumMag_D64C(mlib_d64 *min, const mlib_d64 *x, 
  mlib_s32 m, mlib_s32 n);
```

**Description**  
Each of these functions finds the first element with the minimum magnitude in a complex 
matrix, then puts the real and imaginary parts of it into min[0] and min[1], respectively.

**Parameters**  
Each of the functions takes the following arguments:

- **min**  
  Pointer to the first element with the minimum magnitude.

- **x**  
  Pointer to the first element of the source matrix.

- **m**  
  Number of rows in the source matrix.

- **n**  
  Number of columns in the source matrix.

**Return Values**  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_MatrixMaximumMag_U8C(3MLIB), mlib_VectorMaximumMag_U8C(3MLIB), 
mlib_VectorMinimumMag_U8C(3MLIB), attributes(5)
mlib_MatrixMinimum_U8, mlib_MatrixMinimum_S8, mlib_MatrixMinimum_S16, 
mlib_MatrixMinimum_S32, mlib_MatrixMinimum_F32, mlib_MatrixMinimum_D64 – find 
the minimum value in a matrix

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMinimum_U8(mlib_u8 *min, const mlib_u8 *x, 
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimum_S8(mlib_s8 *min, const mlib_s8 *x, 
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimum_S16(mlib_s16 *min, const mlib_s16 *x, 
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimum_S32(mlib_s32 *min, const mlib_s32 *x, 
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimum_F32(mlib_f32 *min, const mlib_f32 *x, 
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMinimum_D64(mlib_d64 *min, const mlib_d64 *x, 
mlib_s32 m, mlib_s32 n);

Description

Each of these functions finds the minimum value of all elements in a matrix.

It uses the following equation:

\[
min[0] = \text{MIN}\{ x[i] \ i = 0, 1, \ldots, (m*n - 1) \}
\]

Parameters

Each of the functions takes the following arguments:

- \( min \) Pointer to the minimum value.
- \( x \) Pointer to the first element of the source matrix.
- \( m \) Number of rows in the source matrix.
- \( n \) Number of columns in the source matrix.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_MatrixMaximum_U8(3MLIB)
mlib_VectorMaximum_U8(3MLIB)
mlib_VectorMinimum_U8(3MLIB)
attributes(5)
mlib_MatrixMulShift_S16_S16_Mod, mlib_MatrixMulShift_S16_S16_Sat, mlib_MatrixMulShift_S16C_S16C_Mod, mlib_MatrixMulShift_S16C_S16C_Sat – matrix multiplication plus shifting

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMulShift_S16_S16_Mod(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
mlib_s32 l, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_MatrixMulShift_S16_S16_Sat(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
mlib_s32 l, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_MatrixMulShift_S16C_S16C_Mod(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
mlib_s32 l, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_MatrixMulShift_S16C_S16C_Sat(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m,
mlib_s32 l, mlib_s32 n, mlib_s32 shift);

Description

Each of these functions performs a multiplication of two matrices and shifts the result.

For real data, the following equation is used:

\[
\begin{align*}
    z[i*n + j] &= \left\{ \sum_{k=0}^{l-1} (x[i*l + k] * y[k*n + j]) \right\} * 2^{(\text{shift})} \\
\end{align*}
\]

where \( i = 0, 1, \ldots, (m - 1) \); \( j = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
    z[2*(i*n + j)] &= \left\{ \sum_{k=0}^{l-1} (xR*yR - xI*yI) \right\} * 2^{(\text{shift})} \\
    z[2*(i*n + j) + 1] &= \left\{ \sum_{k=0}^{l-1} (xR*yI + xI*yR) \right\} * 2^{(\text{shift})} \\
\end{align*}
\]

where

\[
\begin{align*}
    xR &= x[2*(i*l + k)] \\
    xI &= x[2*(i*l + k) + 1] \\
    yR &= y[2*(k*n + j)] \\
    yI &= y[2*(k*n + j) + 1] \\
    \end{align*}
\]

\( i = 0, 1, \ldots, (m - 1) \)
\( j = 0, 1, \ldots, (n - 1) \)

Parameters

Each of the functions takes the following arguments:
z  Pointer to the first element of the result matrix, in row major order.
x  Pointer to the first element of the first matrix, in row major order.
y  Pointer to the first element of the second matrix, in row major order.
m  Number of rows in the first matrix. \( m > 0 \).
l  Number of columns in the first matrix, and the number of rows in the second matrix. \( l > 0 \).
n  Number of columns in the second matrix. \( n > 0 \).
shift  Right shifting factor. \( 1 \leq \text{shift} \leq 16 \).

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_MatrixMul_U8_U8_Mod(3MLIB), attributes(5)
REFERENCE

Multimedia Library Functions - Part 4
mlib_MatrixMulSShift_U8_Mod, mlib_MatrixMulSShift_U8_Sat,
mlib_MatrixMulSShift_U8C_Mod, mlib_MatrixMulSShift_U8C_Sat,
mlib_MatrixMulSShift_S8_Mod, mlib_MatrixMulSShift_S8_Sat,
mlib_MatrixMulSShift_S8C_Mod, mlib_MatrixMulSShift_S8C_Sat,
mlib_MatrixMulSShift_S16_Mod, mlib_MatrixMulSShift_S16_Sat,
mlib_MatrixMulSShift_S16C_Mod, mlib_MatrixMulSShift_S16C_Sat,
mlib_MatrixMulSShift_S32_Mod, mlib_MatrixMulSShift_S32_Sat – matrix
multiplication by scalar plus shifting, in place

Synopsis

c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMulSShift_U8_Mod(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_U8_Sat(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_U8C_Mod(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_U8C_Sat(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8_Mod(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8_Sat(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8C_Mod(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8C_Sat(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16_Mod(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16_Sat(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16C_Mod(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16C_Sat(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S32_Mod(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S32_Sat(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_MatrixMulSShift_S32C_Mod(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_MatrixMulSShift_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n, mlib_s32 shift);

Description
Each of these functions performs an in-place multiplication of a matrix with a scalar and shifts the result.

For real data, the following equation is used:

\[ xz[i] = c[0] \times xz[i] \times 2^{-shift} \]

where \( i = 0, 1, \ldots, (m \times n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}

tmp &= xz[2*i] \\
xz[2*i] &= (c[0] \times tmp - c[1] \times xz[2*i + 1]) \times 2^{-shift} \\
xz[2*i + 1] &= (c[1] \times tmp + c[0] \times xz[2*i + 1]) \times 2^{-shift}
\end{align*}
\]

where \( i = 0, 1, \ldots, (m \times n - 1) \).

The ranges of valid \( shift \) are:

\( 1 \leq shift \leq 8 \) for U8, S8, U8C, S8C types
\( 1 \leq shift \leq 16 \) for S16, S16C types
\( 1 \leq shift \leq 31 \) for S32, S32C types

Parameters
Each of the functions takes the following arguments:

- \( xz \) Pointer to the source and destination matrix.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( m \) Number of rows in each matrix.
- \( n \) Number of columns in each matrix.
- \( shift \) Right shifting factor.

Return Values
Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

Attributes
See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

Multimedia Library Functions - Part 4 751
See Also  \texttt{mlib\_MatrixMulSShift\_U8\_Mod(3MLIB)}, \texttt{attributes(5)}
mlib_MatrixMulSShift_U8_U8_Mod, mlib_MatrixMulSShift_U8_U8_Sat,
mlib_MatrixMulSShift_U8C_U8C_Mod, mlib_MatrixMulSShift_U8C_U8C_Sat,
mlib_MatrixMulSShift_S8_S8_Mod, mlib_MatrixMulSShift_S8_S8_Sat,
mlib_MatrixMulSShift_S8C_S8C_Mod, mlib_MatrixMulSShift_S8C_S8C_Sat,
mlib_MatrixMulSShift_S16_S16_Mod, mlib_MatrixMulSShift_S16_S16_Sat,
mlib_MatrixMulSShift_S16C_S16C_Mod, mlib_MatrixMulSShift_S16C_S16C_Sat,
mlib_MatrixMulSShift_S32_S32_Mod, mlib_MatrixMulSShift_S32_S32_Sat,
mlib_MatrixMulSShift_S32C_S32C_Mod, mlib_MatrixMulSShift_S32C_S32C_Sat – matrix
multiplication by scalar plus shifting

**Synopsis**

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMulSShift_U8_U8_Mod(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_U8_U8_Sat(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_U8C_U8C_Mod(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_U8C_U8C_Sat(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8_S8_Rem(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8_S8_Sat(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8C_S8C_Mod(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S8C_S8C_Sat(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16_S16_Mod(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16_S16_Sat(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16C_S16C_Mod(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S16C_S16C_Sat(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S32_S32_Mod(mlib_s32 *z,
const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);

mlib_status mlib_MatrixMulSShift_S32_S32_Sat(mlib_s32 *z,
const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n,
mlib_s32 shift);
Each of these functions performs a multiplication of a matrix with a scalar and shifts the result.

For real data, the following equation is used:

\[ z[i] = c[0] \times x[i] \times 2^{\text{shift}} \]

where \( i = 0, 1, \ldots, (m\times n - 1) \).

For complex data, the following equation is used:

\[ z[2i] = (c[0] \times x[2i] - c[1] \times x[2i + 1]) \times 2^{\text{shift}} \]
\[ z[2i + 1] = (c[1] \times x[2i] + c[0] \times x[2i + 1]) \times 2^{\text{shift}} \]

where \( i = 0, 1, \ldots, (m\times n - 1) \).

The ranges of valid shift are:

\[ 1 \leq \text{shift} \leq 8 \] for U8, S8, S16C, S8C types
\[ 1 \leq \text{shift} \leq 16 \] for S16, S16C types
\[ 1 \leq \text{shift} \leq 31 \] for S32, S32C types

Parameters

Each of the functions takes the following arguments:

\( z \)  
Pointer to the destination matrix.

\( x \)  
Pointer to the source matrix.
c
Pointer to the source scalar. When the function is used with complex data types, c[0]
contains the scalar for the real part, and c[1] contains the scalar for the imaginary
part.

m
Number of rows in each matrix.
n
Number of columns in each matrix.

shift
Right shifting factor.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_MatrixMulSShift_U8_U8_Mod(3MLIB), attributes(5)
Name

mlib_MatrixMulS_U8_Mod, mlib_MatrixMulS_U8_Sat, mlib_MatrixMulS_U8C_Mod,
mlib_MatrixMulS_U8C_Sat, mlib_MatrixMulS_S8_Mod, mlib_MatrixMulS_S8_Sat,
mlib_MatrixMulS_S8C_Mod, mlib_MatrixMulS_S8C_Sat, mlib_MatrixMulS_S16_Mod,
mlib_MatrixMulS_S16_Sat, mlib_MatrixMulS_S16C_Mod, mlib_MatrixMulS_S16C_Sat,
mlib_MatrixMulS_S32_Mod, mlib_MatrixMulS_S32_Sat, mlib_MatrixMulS_S32C_Mod,
mlib_MatrixMulS_S32C_Sat – matrix multiplication by scalar, in place

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMulS_U8_Mod(mlib_u8 *xz, const mlib_u8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_U8_Sat(mlib_u8 *xz, const mlib_u8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_U8C_Mod(mlib_u8 *xz, const mlib_u8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_U8C_Sat(mlib_u8 *xz, const mlib_u8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8_Mod(mlib_s8 *xz, const mlib_s8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8_Sat(mlib_s8 *xz, const mlib_s8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8C_Mod(mlib_s8 *xz, const mlib_s8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8C_Sat(mlib_s8 *xz, const mlib_s8 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_Mod(mlib_s16 *xz, const mlib_s16 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_Sat(mlib_s16 *xz, const mlib_s16 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16C_Mod(mlib_s16 *xz, const mlib_s16 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16C_Sat(mlib_s16 *xz, const mlib_s16 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32_Mod(mlib_s32 *xz, const mlib_s32 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32_Sat(mlib_s32 *xz, const mlib_s32 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32C_Mod(mlib_s32 *xz, const mlib_s32 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c,
mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c,
               mlib_s32 m, mlib_s32 n);

Description Each of these functions performs an in-place multiplication of a scalar to a matrix.

For real data, the following equation is used:

\[ xz[i] = c[0]*xz[i] \]

where \( i = 0, 1, \ldots, (m*n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
\text{tmp} &= xz[2*i] \\
xz[2*i] &= c[0]*\text{tmp} - c[1]*xz[2*i + 1] \\
xz[2*i + 1] &= c[1]*\text{tmp} + c[0]*xz[2*i + 1]
\end{align*}
\]

where \( i = 0, 1, \ldots, (m*n - 1) \).

Parameters Each of the functions takes the following arguments:

- \( xz \) Pointer to the source and destination matrix.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( m \) Number of rows in each matrix.
- \( n \) Number of columns in each matrix.

Return Values Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also \texttt{mlib_MatrixMulS_U8_U8_Mod(3MLIB)}, attributes(5)
Name  mlib_MatrixMulS_U8_U8_Mod, mlib_MatrixMulS_U8_U8_Sat, 
mlib_MatrixMulS_U8C_U8C_Mod, mlib_MatrixMulS_U8C_U8C_Sat, 
mlib_MatrixMulS_S8_S8_Mod, mlib_MatrixMulS_S8_S8_Sat, 
mlib_MatrixMulS_S8C_S8C_Mod, mlib_MatrixMulS_S8C_S8C_Sat, 
mlib_MatrixMulS_S16_U8_Mod, mlib_MatrixMulS_S16_U8_Sat, 
mlib_MatrixMulS_S16_S8_Mod, mlib_MatrixMulS_S16_S8_Sat, 
mlib_MatrixMulS_S16C_S16_Mod, mlib_MatrixMulS_S16C_S16_Sat, 
mlib_MatrixMulS_S32_S16_Mod, mlib_MatrixMulS_S32_S16_Sat, 
mlib_MatrixMulS_S32C_S32_Mod, mlib_MatrixMulS_S32C_S32_Sat — matrix 
multiplication by scalar

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMulS_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x, 
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x, 
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x, 
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x, 
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x, 
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x, 
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x, 
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x, 
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x, 
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x, 
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x, 
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, 
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x, 
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x, 
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16C_S16_Mod(mlib_s16 *z, const mlib_s16 *x, 
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16C_S16_Sat(mlib_s16 *z, const mlib_s16 *x, 
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, 
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, 
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixMulS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

Description

Each of these functions multiplies a matrix by a scalar.

For real data, the following equation is used:

\[ z[i] = c[0] \times x[i] \]
where \( i = 0, 1, \ldots, (m*n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
z[2*i + 1] &= c[1]*x[2*i] + c[0]*x[2*i + 1]
\end{align*}
\]

where \( i = 0, 1, \ldots, (m*n - 1) \).

**Parameters**
Each of the functions takes the following arguments:

- \( z \) Pointer to the destination matrix.
- \( x \) Pointer to the source matrix.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( m \) Number of rows in each matrix.
- \( n \) Number of columns in each matrix.

**Return Values**
Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

**Attributes**
See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
\texttt{mlib_MatrixMulS_U8_U8_Mod(3MLIB)}, \texttt{attributes(5)}
Name
mlib_MatrixMul_U8_U8_Mod, mlib_MatrixMul_U8_U8_Sat,
mlib_MatrixMul_U8C_U8C_Mod, mlib_MatrixMul_U8C_U8C_Sat,
mlib_MatrixMul_S8_S8_Mod, mlib_MatrixMul_S8_S8_Sat,
mlib_MatrixMul_S8C_S8C_Mod, mlib_MatrixMul_S8C_S8C_Sat,
mlib_MatrixMul_S16_U8_Mod, mlib_MatrixMul_S16_U8_Sat,
mlib_MatrixMul_S16_S8_Mod, mlib_MatrixMul_S16_S8_Sat,
mlib_MatrixMul_S16_S16_Mod, mlib_MatrixMul_S16_S16_Sat,
mlib_MatrixMul_S16C_U8C_Mod, mlib_MatrixMul_S16C_U8C_Sat,
mlib_MatrixMul_S16C_S8C_Mod, mlib_MatrixMul_S16C_S8C_Sat,
mlib_MatrixMul_S16C_S16C_Mod, mlib_MatrixMul_S16C_S16C_Sat,
mlib_MatrixMul_S32_S16_Mod, mlib_MatrixMul_S32_S16_Sat,
mlib_MatrixMul_S32_S32_Mod, mlib_MatrixMul_S32_S32_Sat–matrix
multiplication

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixMul_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
  const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
  const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16C_U8C_Mod(mlib_s16c *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16C_U8C_Sat(mlib_s16c *z, const mlib_u8 *x,
  const mlib_u8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16C_S8C_Mod(mlib_s16c *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16C_S8C_Sat(mlib_s16c *z, const mlib_s8 *x,
  const mlib_s8 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16C_S16C_Mod(mlib_s16c *z, const mlib_s16 *x,
  const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S16C_S16C_Sat(mlib_s16c *z, const mlib_s16 *x,
  const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
  const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
  const mlib_s16 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
  const mlib_s32 *y, mlib_s32 m, mlib_s32 l, mlib_s32 n);
mlib_status mlib_MatrixMul_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
Each of these functions performs matrix multiplication of the first matrix to the second matrix or the first complex matrix to the second complex matrix.

For real data, the following equation is used:

\[
\text{Description}
\]

For real data, the following equation is used:
\[ z[i*n + j] = \sum_{k=0}^{l-1} (x[i*l + k] * y[k*n + j]) \]

where \( i = 0, 1, \ldots, (m - 1) \); \( j = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[ z[2*(i*n + j)] = \sum_{k=0}^{l-1} (xR*yR - xI*yI) \]
\[ z[2*(i*n + j) + 1] = \sum_{k=0}^{l-1} (xR*yI + xI*yR) \]

where

\( xR = x[2*(i*l + k)] \)
\( xI = x[2*(i*l + k) + 1] \)
\( yR = y[2*(k*n + j)] \)
\( yI = y[2*(k*n + j) + 1] \)
\( i = 0, 1, \ldots, (m - 1) \)
\( j = 0, 1, \ldots, (n - 1) \)

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the destination matrix.
- \( x \) Pointer to the first source matrix.
- \( y \) Pointer to the second source matrix.
- \( m \) Number of rows in the first matrix.
- \( l \) Number of columns in the first matrix, and number of rows in the second matrix.
- \( n \) Number of columns in the second matrix.

**Return Values** Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_MatrixMulShift_S16_S16_Mod(3MLIB), attributes(5)
**Name**
mlib\_MatrixScale\_U8\_Mod, mlib\_MatrixScale\_U8\_Sat, mlib\_MatrixScale\_U8\_C\_Mod,
mlib\_MatrixScale\_U8\_C\_Sat, mlib\_MatrixScale\_S8\_Mod, mlib\_MatrixScale\_S8\_Sat,
mlib\_MatrixScale\_S8\_C\_Mod, mlib\_MatrixScale\_S8\_C\_Sat, mlib\_MatrixScale\_S16\_Mod,
mlib\_MatrixScale\_S16\_Sat, mlib\_MatrixScale\_S16\_C\_Mod, mlib\_MatrixScale\_S16\_C\_Sat,
mlib\_MatrixScale\_S32\_Mod, mlib\_MatrixScale\_S32\_Sat, mlib\_MatrixScale\_S32\_C\_Mod,
mlib\_MatrixScale\_S32\_C\_Sat – matrix linear scaling, in place

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib\_status mlib\_MatrixScale\_U8\_Mod(mlib\_u8 *\_xz, const mlib\_u8 *\_a,
const mlib\_u8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_U8\_Sat(mlib\_u8 *\_xz, const mlib\_u8 *\_a,
const mlib\_u8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_U8\_C\_Mod(mlib\_u8 *\_xz, const mlib\_u8 *\_a,
const mlib\_u8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_U8\_C\_Sat(mlib\_u8 *\_xz, const mlib\_u8 *\_a,
const mlib\_u8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S8\_Mod(mlib\_s8 *\_xz, const mlib\_s8 *\_a,
const mlib\_s8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S8\_Sat(mlib\_s8 *\_xz, const mlib\_s8 *\_a,
const mlib\_s8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S8\_C\_Mod(mlib\_s8 *\_xz, const mlib\_s8 *\_a,
const mlib\_s8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S8\_C\_Sat(mlib\_s8 *\_xz, const mlib\_s8 *\_a,
const mlib\_s8 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S16\_Mod(mlib\_s16 *\_xz, const mlib\_s16 *\_a,
const mlib\_s16 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S16\_Sat(mlib\_s16 *\_xz, const mlib\_s16 *\_a,
const mlib\_s16 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S16\_C\_Mod(mlib\_s16 *\_xz, const mlib\_s16 *\_a,
const mlib\_s16 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S16\_C\_Sat(mlib\_s16 *\_xz, const mlib\_s16 *\_a,
const mlib\_s16 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S32\_Mod(mlib\_s32 *\_xz, const mlib\_s32 *\_a,
const mlib\_s32 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S32\_Sat(mlib\_s32 *\_xz, const mlib\_s32 *\_a,
const mlib\_s32 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S32\_C\_Mod(mlib\_s32 *\_xz, const mlib\_s32 *\_a,
const mlib\_s32 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);

mlib\_status mlib\_MatrixScale\_S32\_C\_Sat(mlib\_s32 *\_xz, const mlib\_s32 *\_a,
const mlib\_s32 *\_b, mlib\_s32 \_m, mlib\_s32 \_n);
mlib_status mlib_MatrixScale_S32C_Sat(mlib_s32 *xz, const mlib_s32 *a, const mlib_s32 *b, mlib_s32 m, mlib_s32 n);

Description  Each of these functions performs an in-place multiplication of a matrix by a scalar and then adds an offset.

For real data, the following equation is used:

\[ xz[i] = a[0] \times xz[i] + b[0] \]

where \( i = 0, 1, \ldots, (m \times n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
    \text{tmp} &= xz[2*i] \\
    xz[2*i] &= a[0] \times \text{tmp} - a[1] \times xz[2*i + 1] + b[0] \\
    xz[2*i + 1] &= a[1] \times \text{tmp} + a[0] \times xz[2*i + 1] + b[1]
\end{align*}
\]

where \( i = 0, 1, \ldots, (m \times n - 1) \).

Parameters  Each of the functions takes the following arguments:

- \( xz \): Pointer to the source and destination matrix.
- \( a \): Pointer to the source scaling factor. When the function is used with complex data types, \( a[0] \) contains the scalar for the real part, and \( a[1] \) contains the scalar for the imaginary part.
- \( b \): Pointer to the source offset. When the function is used with complex data types, \( b[0] \) contains the offset for the real part, and \( b[1] \) contains the offset for the imaginary part.
- \( m \): Number of rows in the matrix.
- \( n \): Number of columns in the matrix.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_MatrixScale_U8_U8_Mod(3MLIB), attributes(5)
Name

mlib_MatrixScale_U8_U8_Mod, mlib_MatrixScale_U8_U8_Sat,
mlib_MatrixScale_U8C_U8C_Mod, mlib_MatrixScale_U8C_U8C_Sat,
mlib_MatrixScale_S8_S8_Mod, mlib_MatrixScale_S8_S8_Sat,
mlib_MatrixScale_S8C_S8C_Mod, mlib_MatrixScale_S8C_S8C_Sat,
mlib_MatrixScale_S16_U8_Mod, mlib_MatrixScale_S16_U8_Sat,
mlib_MatrixScale_S16_S8_Mod, mlib_MatrixScale_S16_S8_Sat,
mlib_MatrixScale_S16_S16_Mod, mlib_MatrixScale_S16_S16_Sat,
mlib_MatrixScale_S16C_U8C_Mod, mlib_MatrixScale_S16C_U8C_Sat,
mlib_MatrixScale_S16C_S8C_Mod, mlib_MatrixScale_S16C_S8C_Sat,
mlib_MatrixScale_S16C_S16C_Mod, mlib_MatrixScale_S16C_S16C_Sat,
mlib_MatrixScale_S32_S16_Mod, mlib_MatrixScale_S32_S16_Sat,
mlib_MatrixScale_S32_S32_Mod, mlib_MatrixScale_S32_S32_Sat – matrix linear
scaling

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixScale_U8_U8_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_U8_U8_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_U8C_U8C_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_U8C_U8C_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S8_S8_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S8_S8_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S8C_S8C_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S8C_S8C_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16_U8_Mod(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16_U8_Sat(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16_S8_Mod(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16_S8_Sat(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16_S16_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16_S16_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16C_U8C_Mod(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16C_U8C_Sat(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16C_S8C_Mod(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16C_S8C_Sat(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16C_S16C_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S16C_S16C_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S32_S16_Mod(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S32_S16_Sat(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixScale_S32_S32_Mod(mlib_s32 *z,  
    const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b,  
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S32_S32_Sat(mlib_s32 *z,  
    const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b,  
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S32C_S16C_Mod(mlib_s32 *z,  
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,  
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S32C_S16C_Sat(mlib_s32 *z,  
    const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b,  
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S32C_S32C_Mod(mlib_s32 *z,  
    const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b,  
    mlib_s32 m, mlib_s32 n);

mlib_status mlib_MatrixScale_S32C_S32C_Sat(mlib_s32 *z,  
    const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b,  
    mlib_s32 m, mlib_s32 n);

Description Each of these functions multiplies a matrix by a scalar and then adds an offset.

For real data, the following equation is used:

\[ z[i] = a[0] \cdot x[i] + b[0] \]

where \( i = 0, 1, \ldots, (m \cdot n - 1) \).

For complex data, the following equation is used:

\[ z[2 \cdot i] = a[0] \cdot x[2 \cdot i] - a[1] \cdot x[2 \cdot i + 1] + b[0] \]
\[ z[2 \cdot i + 1] = a[1] \cdot x[2 \cdot i] + a[0] \cdot x[2 \cdot i + 1] + b[1] \]

where \( i = 0, 1, \ldots, (m \cdot n - 1) \).

Parameters Each of the functions takes the following arguments:

\( z \) Pointer to the destination matrix.

\( x \) Pointer to the source matrix.

\( a \) Pointer to the source scaling factor. When the function is used with complex data types, \( a[0] \) contains the scalar for the real part, and \( a[1] \) contains the scalar for the imaginary part.

\( b \) Pointer to the source offset. When the function is used with complex data types, \( b[0] \) contains the offset for the real part, and \( b[1] \) contains the offset for the imaginary part.

\( m \) Number of rows in each matrix.

\( n \) Number of columns in each matrix.
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_MatrixScale_U8_U8_Mod(3MLIB), attributes(5)
mlib_MatrixSubS_U8_Mod, mlib_MatrixSubS_U8_Sat, mlib_MatrixSubS_U8C_Mod,
mlib_MatrixSubS_U8C_Sat, mlib_MatrixSubS_S8_Mod, mlib_MatrixSubS_S8_Sat,
mlib_MatrixSubS_S8C_Mod, mlib_MatrixSubS_S8C_Sat, mlib_MatrixSubS_S16_Mod,
mlib_MatrixSubS_S16_Sat, mlib_MatrixSubS_S16C_Mod, mlib_MatrixSubS_S16C_Sat,
mlib_MatrixSubS_S32_Mod, mlib_MatrixSubS_S32_Sat, mlib_MatrixSubS_S32C_Mod,
mlib_MatrixSubS_S32C_Sat – matrix subtraction from scalar, in place

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixSubS_U8_Mod(mlib_u8 *xZ, const mlib_u8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_U8_Sat(mlib_u8 *xZ, const mlib_u8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_U8C_Mod(mlib_u8 *xZ, const mlib_u8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_U8C_Sat(mlib_u8 *xZ, const mlib_u8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8_Mod(mlib_s8 *xZ, const mlib_s8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8_Sat(mlib_s8 *xZ, const mlib_s8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8C_Mod(mlib_s8 *xZ, const mlib_s8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8C_Sat(mlib_s8 *xZ, const mlib_s8 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_Mod(mlib_s16 *xZ, const mlib_s16 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_Sat(mlib_s16 *xZ, const mlib_s16 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_Mod(mlib_s16 *xZ, const mlib_s16 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_Sat(mlib_s16 *xZ, const mlib_s16 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32_Mod(mlib_s32 *xZ, const mlib_s32 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32_Sat(mlib_s32 *xZ, const mlib_s32 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32C_Mod(mlib_s32 *xZ, const mlib_s32 *c,
    mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

**Description**  Each of these functions performs an in-place subtraction of a matrix from a scalar.

For real data, the following equation is used:

\[ xz[i] = c[0] - xz[i] \]

where \( i = 0, 1, \ldots, (m*n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
\end{align*}
\]

where \( i = 0, 1, \ldots, (m*n - 1) \).

**Parameters**  Each of the functions takes the following arguments:

- **xz**: Pointer to the source and destination matrix.
- **c**: Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- **m**: Number of rows in the matrices.
- **n**: Number of columns in the matrices.

**Return Values**  Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_MatrixSubS_U8_U8_Mod(3MLIB), attributes(5)
Name  mlib_MatrixSubS_U8_U8_Mod, mlib_MatrixSubS_U8_U8_Sat,
mlib_MatrixSubS_U8C_U8C_Mod, mlib_MatrixSubS_U8C_U8C_Sat,
mlib_MatrixSubS_S8_S8_Mod, mlib_MatrixSubS_S8_S8_Sat,
mlib_MatrixSubS_S8C_S8C_Mod, mlib_MatrixSubS_S8C_S8C_Sat,
mlib_MatrixSubS_S16_U8_Mod, mlib_MatrixSubS_S16_U8_Sat,
mlib_MatrixSubS_S16_S8_Mod, mlib_MatrixSubS_S16_S8_Sat,
mlib_MatrixSubS_S16_S16_Mod, mlib_MatrixSubS_S16_S16_Sat,
mlib_MatrixSubS_S16C_U8C_Mod, mlib_MatrixSubS_S16C_U8C_Sat,
mlib_MatrixSubS_S16C_S8C_Mod, mlib_MatrixSubS_S16C_S8C_Sat,
mlib_MatrixSubS_S16C_S16C_Mod, mlib_MatrixSubS_S16C_S16C_Sat,
mlib_MatrixSubS_S32_S16_Mod, mlib_MatrixSubS_S32_S16_Sat,
mlib_MatrixSubS_S32_S32_Mod, mlib_MatrixSubS_S32_S32_Sat—matrix
subtraction from scalar

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixSubS_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_S8C_Mod(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_S8C_Sat(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSubS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 m, mlib_s32 n);

Description Each of these functions subtracts a matrix from a scalar.

For real data, the following equation is used:

\[ z[i] = c[0] - x[i] \]
where \( i = 0, 1, \ldots, (m*n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
    z[2*i] &= c[0] - x[2*i] \\
\end{align*}
\]

where \( i = 0, 1, \ldots, (m*n - 1) \).

**Parameters**

Each of the functions takes the following arguments:

- **\( z \)**  Pointer to the destination matrix.
- **\( x \)**  Pointer to the source matrix.
- **\( c \)**  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- **\( m \)**  Number of rows in the matrices.
- **\( n \)**  Number of columns in the matrices.

**Return Values**

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_MatrixSubS_U8_U8_Mod(3MLIB), attributes(5)
Name  mlib_MatrixSub_U8_Mod, mlib_MatrixSub_U8_Sat, mlib_MatrixSub_U8C_Mod,  
      mlib_MatrixSub_U8C_Sat, mlib_MatrixSub_S8_Mod, mlib_MatrixSub_S8_Sat,  
      mlib_MatrixSub_S8C_Mod, mlib_MatrixSub_S8C_Sat, mlib_MatrixSub_S16_Mod,  
      mlib_MatrixSub_S16_Sat, mlib_MatrixSub_S16C_Mod, mlib_MatrixSub_S16C_Sat,  
      mlib_MatrixSub_S32_Mod, mlib_MatrixSub_S32_Sat, mlib_MatrixSub_S32C_Mod,  
      mlib_MatrixSub_S32C_Sat – matrix subtraction, in place

Synopsis  cc [ flag... ] file... -lmlib [ library... ]  
          #include <mlib.h>

      mlib_status mlib_MatrixSub_U8_Mod(mlib_u8 *xz,  
             const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_U8_Sat(mlib_u8 *xz,  
             const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_U8C_Mod(mlib_u8 *xz,  
             const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_U8C_Sat(mlib_u8 *xz,  
             const mlib_u8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S8_Mod(mlib_s8 *xz,  
             const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S8_Sat(mlib_s8 *xz,  
             const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S8C_Mod(mlib_s8 *xz,  
             const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S8C_Sat(mlib_s8 *xz,  
             const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S16_Mod(mlib_s16 *xz,  
             const mlib_s16 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S16_Sat(mlib_s16 *xz,  
             const mlib_s16 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S16C_Mod(mlib_s16 *xz,  
             const mlib_s16 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S16C_Sat(mlib_s16 *xz,  
             const mlib_s16 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S32_Mod(mlib_s32 *xz,  
             const mlib_s32 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S32_Sat(mlib_s32 *xz,  
             const mlib_s32 *y, mlib_s32 m, mlib_s32 n);

      mlib_status mlib_MatrixSub_S32C_Mod(mlib_s32 *xz,  
             const mlib_s32 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 m, mlib_s32 n);

Description  Each of these functions performs an in-place subtraction of the second matrix from the first matrix.

It uses the following equation:

\[ xz[i] = xz[i] - y[i] \]

where \( i = 0, 1, \ldots, (m*n - 1) \) for real data; \( i = 0, 1, \ldots, (m*n*2 - 1) \) for complex data.

Parameters  Each of the functions takes the following arguments:

- \( xz \)  Pointer to the first source and destination matrix.
- \( y \)  Pointer to the second source matrix.
- \( m \)  Number of rows in the matrices.
- \( n \)  Number of columns in the matrices.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_MatrixSub_U8_U8_Mod(3MLIB), attributes(5)
mlib_MatrixSub_U8_U8_Mod, mlib_MatrixSub_U8_U8_Sat,
mlib_MatrixSub_U8C_U8C_Mod, mlib_MatrixSub_U8C_U8C_Sat,
mlib_MatrixSub_S8_S8_Mod, mlib_MatrixSub_S8_S8_Sat,
mlib_MatrixSub_S8C_S8C_Mod, mlib_MatrixSub_S8C_S8C_Sat,
mlib_MatrixSub_S16_U8_Mod, mlib_MatrixSub_S16_U8_Sat,
mlib_MatrixSub_S16_S8_Mod, mlib_MatrixSub_S16_S8_Sat,
mlib_MatrixSub_S16_S16_Mod, mlib_MatrixSub_S16_S16_Sat,
mlib_MatrixSub_S16C_S16C_Mod, mlib_MatrixSub_S16C_S16C_Sat,
mlib_MatrixSub_S32_S16_Mod, mlib_MatrixSub_S32_S16_Sat,
mlib_MatrixSub_S32_S32_Mod, mlib_MatrixSub_S32_S32_Sat

Name
mlib_MatrixSub_U8_U8_Mod, mlib_MatrixSub_U8_U8_Sat,
mlib_MatrixSub_U8C_U8C_Mod, mlib_MatrixSub_U8C_U8C_Sat,
mlib_MatrixSub_S8_S8_Mod, mlib_MatrixSub_S8_S8_Sat,
mlib_MatrixSub_S8C_S8C_Mod, mlib_MatrixSub_S8C_S8C_Sat,
mlib_MatrixSub_S16_U8_Mod, mlib_MatrixSub_S16_U8_Sat,
mlib_MatrixSub_S16_S8_Mod, mlib_MatrixSub_S16_S8_Sat,
mlib_MatrixSub_S16_S16_Mod, mlib_MatrixSub_S16_S16_Sat,
mlib_MatrixSub_S16C_S16C_Mod, mlib_MatrixSub_S16C_S16C_Sat,
mlib_MatrixSub_S32_S16_Mod, mlib_MatrixSub_S32_S16_Sat,
mlib_MatrixSub_S32_S32_Mod, mlib_MatrixSub_S32_S32_Sat

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixSub_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
                                      const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
                                      const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
                                      const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
                                      const mlib_u8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
                                      const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
                                      const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
                                      const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
                                      const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S16_S16_Mod(mlib_s16 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S16_S16_Sat(mlib_s16 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S16_S8_Mod(mlib_s16 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S16_S8_Sat(mlib_s16 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S16C_S16C_Mod(mlib_s16 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S16C_S16C_Sat(mlib_s16 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S32_S32_Mod(mlib_s32 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixSub_S32_S32_Sat(mlib_s32 *z, const mlib_u8 *x,
                                       const mlib_s8 *y, mlib_s32 m, mlib_s32 n);

matrix subtraction
Each of these functions subtracts the second matrix from the first matrix.

It uses the following equation:

\[ z[i] = x[i] - y[i] \]

where \( i = 0, 1, \ldots, (m*n - 1) \) for real data; \( i = 0, 1, \ldots, (m*n*2 - 1) \) for complex data.

Description

Each of these functions subtracts the second matrix from the first matrix.

It uses the following equation:

\[ z[i] = x[i] - y[i] \]

where \( i = 0, 1, \ldots, (m*n - 1) \) for real data; \( i = 0, 1, \ldots, (m*n*2 - 1) \) for complex data.
Parameters  Each of the functions takes the following arguments:

- \( z \)  Pointer to the destination matrix.
- \( x \)  Pointer to the first source matrix.
- \( y \)  Pointer to the second source matrix.
- \( m \)  Number of rows in the matrices.
- \( n \)  Number of columns in the matrices.

Return Values  Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

Attributes  See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  \texttt{mlib_MatrixSub_U8_U8_Mod(3MLIB), attributes(5)}
Each of these functions performs an in-place transpose of a square matrix.

For real data, the following pseudo code applies:

```c
for (i = 1; i < mn; i++) {
    for (j = 0; j < i; j++) {
        tmp = xz[i*mn + j];
        xz[i*mn + j] = xz[j*mn + i];
        xz[j*mn + i] = tmp;
    }
}
```

For complex data, the following pseudo code applies:

```c
for (i = 1; i < mn; i++) {
    for (j = 0; j < i; j++) {
        tmp0 = xz[2*(i*mn + j)];
        tmp1 = xz[2*(i*mn + j) + 1];
        xz[2*(i*mn + j)] = xz[2*(j*mn + i)];
        xz[2*(i*mn + j) + 1] = xz[2*(j*mn + i) + 1];
        xz[2*(j*mn + i)] = tmp0;
        xz[2*(j*mn + i) + 1] = tmp1;
    }
}
```

Each of the functions takes the following arguments:

- `xz` Pointer to the source and destination matrix.
- `mn` Number of rows and columns in the matrix.
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_MatrixTranspose_U8(3MLIB), attributes(5)
mlib_MatrixTranspose_U8_U8, mlib_MatrixTranspose_U8C_U8C,
mlib_MatrixTranspose_S8_S8, mlib_MatrixTranspose_S8C_S8C,
mlib_MatrixTranspose_S16_S16, mlib_MatrixTranspose_S16C_S16C,
mlib_MatrixTranspose_S32_S32, mlib_MatrixTranspose_S32C_S32C – matrix transpose

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixTranspose_U8_U8(mlib_u8 *z,
    const mlib_u8 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_U8C_U8C(mlib_u8 *z,
    const mlib_u8 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_S8_S8(mlib_s8 *z,
    const mlib_s8 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_S8C_S8C(mlib_s8 *z,
    const mlib_s8 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_S16_S16(mlib_s16 *z,
    const mlib_s16 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_S16C_S16C(mlib_s16 *z,
    const mlib_s16 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_S32_S32(mlib_s32 *z,
    const mlib_s32 *x, mlib_s32 m, mlib_s32 n);
mlib_status mlib_MatrixTranspose_S32C_S32C(mlib_s32 *z,
    const mlib_s32 *x, mlib_s32 m, mlib_s32 n);

Description  
Each of these functions computes the transpose of the input matrix.

For real data, the following equation is used:

\[ z[j*m + i] = x[i*n + j] \]

where \( i = 0, 1, \ldots, (m - 1) \); \( j = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[ z[2*(j*m + i)] = x[2*(i*n + j)] \]
\[ z[2*(j*m + i) + 1] = x[2*(i*n + j) + 1] \]

where \( i = 0, 1, \ldots, (m - 1) \); \( j = 0, 1, \ldots, (n - 1) \).

Parameters  
Each of the functions takes the following arguments:

\( z \)  
Pointer to the destination matrix. The output data type must be the same as the input data type.

\( x \)  
Pointer to the source matrix.

\( m \)  
Number of rows in the source matrix.
Number of columns in the source matrix.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_MatrixTranspose_U8_U8(3MLIB), attributes(5)
mlib_MatrixUnit_U8, mlib_MatrixUnit_U8C, mlib_MatrixUnit_S8, mlib_MatrixUnit_S8C, mlib_MatrixUnit_S16, mlib_MatrixUnit_S16C, mlib_MatrixUnit_S32, mlib_MatrixUnit_S32C – Unit matrix generation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_MatrixUnit_U8(mlib_u8 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_U8C(mlib_u8 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_S8(mlib_s8 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_S8C(mlib_s8 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_S16(mlib_s16 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_S16C(mlib_s16 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_S32(mlib_s32 *z, mlib_s32 n);
mlib_status mlib_MatrixUnit_S32C(mlib_s32 *z, mlib_s32 n);

Description

Each of these functions sets the values for a unit matrix.

For real data, the following equation is used:

\[
z[i*n + j] = 1 \text{ if } i == j \\
z[i*n + j] = 0 \text{ if } i != j
\]

where \( i = 0, 1, \ldots, (n - 1) \); \( j = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
z[2*(i*n + j)] = 1 \text{ if } i == j \\
z[2*(i*n + j)] = 0 \text{ if } i != j \\
z[2*(i*n + j) + 1] = 0
\]

where \( i = 0, 1, \ldots, (n - 1) \); \( j = 0, 1, \ldots, (n - 1) \).

Parameters

Each of the functions takes the following arguments:

- \( z \) Pointer to the destination matrix.
- \( n \) Number of rows and columns in the matrix.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>ATTRIBUTE TYPE</td>
<td>ATTRIBUTE VALUE</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
[attributes(5)]
mlib_memcpy – copy a block of bytes

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void *mlib_memcpy(void *dst, const void *src, size_t n);
```

Description

The `mlib_memcpy()` function copies `n` bytes from memory area `src` to `dst`. It returns `dst`. The memory areas may not overlap. Use `mlib_memmove()` if the memory areas do overlap.

This function is a wrapper of the standard C function `memcpy()`.

Parameters

The function takes the following arguments:

- `dst` Pointer to the destination.
- `src` Pointer to the source.
- `n` Number of bytes to be copied.

Return Values

The function returns a pointer to the destination.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_memmove(3MLIB), mlib_memset(3MLIB), memory(3C), attributes(5)`
mlib_memmove(3MLIB)

Name  mlib_memmove – copy a block of bytes

Synopsis  

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void *mlib_memmove(void *dst, const void *src, size_t n);
```

Description  The mlib_memmove() function copies \( n \) bytes from memory area \( src \) to \( dst \). Copying between objects that overlap will take place correctly. It returns \( dst \).

This function is a wrapper of the standard C function memmove().

Parameters  The function takes the following arguments:

- \( dst \)  Pointer to the destination.
- \( src \)  Pointer to the source.
- \( n \)  Number of bytes to be copied.

Return Values  The function returns a pointer to the destination.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_memcpy(3MLIB), mlib_memset(3MLIB), memory(3C), attributes(5)
The `mlib_memset()` function sets the first \( n \) bytes in memory area \( s \) to the value of \( c \) (converted to an unsigned char). It returns \( s \).

This function is a wrapper of the standard C function `memset()`.

### Parameters
The function takes the following arguments:
- \( s \) Pointer to the destination.
- \( c \) Value to set.
- \( n \) Number of bytes to be set.

### Return Values
The function returns a pointer to the destination.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_memcpy(3MLIB)`, `mlib_memmove(3MLIB)`, `memory(3C)`, `attributes(5)`
The `mlib_realloc()` function changes the size of the block pointed to by `ptr` to `size` bytes and returns a pointer to the (possibly moved) block.

This function is a wrapper of the standard C function `realloc()`.

### Parameters
- `size`: New size of the block in bytes.
- `ptr`: Pointer to a block.

### Return Values
The function returns a pointer to the reallocated block if successful. Otherwise it returns a null pointer.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_free(3MLIB)`, `mlib_malloc(3MLIB)`, `malloc(3C)`, `attributes(5)`
The `mlib_SignalADPCM2Bits2Linear()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.723 or G.726 16kbps 2-bit ADPCM to 16-bit linear PCM format.

The function takes the following arguments:

- `pcm` Linear PCM sample array.
- `adpcm` ADPCM code array.
- `state` Internal structure of the codec.
- `n` Number of samples in the source array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalADPCM3Bits2Linear(3MLIB)`
- `mlib_SignalADPCM4Bits2Linear(3MLIB)`
- `mlib_SignalADPCM5Bits2Linear(3MLIB)`
- `mlib_SignalADPCMFree(3MLIB)`
- `mlib_SignalADPCMInit(3MLIB)`
- `mlib_SignalADPCM3Bits2Linear(3MLIB)`
- `mlib_SignalADPCM4Bits2Linear(3MLIB)`
- `mlib_SignalADPCM5Bits2Linear(3MLIB)`
- `attributes(5)`
mlib_SignalADPCM3Bits2Linear

Name
mlib_SignalADPCM3Bits2Linear – adaptive differential pulse code modulation (ADPCM)

Synopsis
cc [ flag... ] file... -tclib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalADPCM3Bits2Linear(mlib_s16 *pcm,
    const mlib_u8 *adpcm, void *state, mlib_s32 n);

Description
The mlib_SignalADPCM3Bits2Linear() function performs adaptive differential pulse code
modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726
specifications. It converts data from G.723 or G.726 24kbps 3-bit ADPCM to 16-bit linear
PCM format.

Parameters
The function takes the following arguments:

- **pcm**       Linear PCM sample array.
- **adpcm**     ADPCM code array.
- **state**     Internal structure of the codec.
- **n**         Number of samples in the source array.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalADPCM2Bits2Linear(3MLIB), mlib_SignalADPCM4Bits2Linear(3MLIB),
mlib_SignalADPCM5Bits2Linear(3MLIB), mlib_SignalADPCMInit(3MLIB),
mlib_SignalADPCM2Bits(3MLIB), mlib_SignalADPCM4Bits(3MLIB),
mlib_SignalADPCM5Bits(3MLIB), mlib_SignalLinear2ADPCM2Bits(3MLIB),
mlib_SignalLinear2ADPCM4Bits(3MLIB), mlib_SignalLinear2ADPCM5Bits(3MLIB), attributes(5)
The mlib_SignalADPCM4Bits2Linear() function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.721 or G.726 32kbps 4-bit ADPCM to 16-bit linear PCM format.

The function takes the following arguments:

- `pcm`: Linear PCM sample array.
- `adpcm`: ADPCM code array.
- `state`: Internal structure of the codec.
- `n`: Number of samples in the source array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also mlib_SignalADPCM2Bits2Linear(3MLIB), mlib_SignalADPCM3Bits2Linear(3MLIB), mlib_SignalADPCM5Bits2Linear(3MLIB), mlib_SignalADPCMFree(3MLIB), mlib_SignalADPCMInit(3MLIB), mlib_SignalLinear2ADPCM2Bits(3MLIB), mlib_SignalLinear2ADPCM3Bits(3MLIB), mlib_SignalLinear2ADPCM4Bits(3MLIB), mlib_SignalLinear2ADPCM5Bits(3MLIB), attributes(5)
# mlib_SignalADPCM5Bits2Linear()

The `mlib_SignalADPCM5Bits2Linear()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from G.723 or G.726 40kbps 5-bit ADPCM to 16-bit linear PCM format.

The function takes the following arguments:

- **pcm**: Linear PCM sample array.
- **adpcm**: ADPCM code array.
- **state**: Internal structure of the codec.
- **n**: Number of samples in the source array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalADPCM2Bits2Linear(3MLIB)`
- `mlib_SignalADPCM3Bits2Linear(3MLIB)`
- `mlib_SignalADPCM4Bits2Linear(3MLIB)`
- `mlib_SignalADPCMFree(3MLIB)`
- `mlib_SignalADPCMInit(3MLIB)`
- `mlib_SignalLinear2ADPCM2Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM3Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM4Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM5Bits(3MLIB)`
- attributes(5)
Name  
mlib_SignalADPCMFree – adaptive differential pulse code modulation (ADPCM)

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalADPCMFree(void *state);

Description  
The mlib_SignalADPCMFree() function frees the internal structure for the codec for functions that perform adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications.

Parameters  
The function takes the following arguments:

state Internal structure of the codec.

Return Values  
None.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalADPCM2Bits2Linear(3MLIB), mlib_SignalADPCM3Bits2Linear(3MLIB), mlib_SignalADPCM4Bits2Linear(3MLIB), mlib_SignalADPCM5Bits2Linear(3MLIB), mlib_SignalADPCMInit(3MLIB), mlib_SignalLinear2ADPCM2Bits(3MLIB), mlib_SignalLinear2ADPCM3Bits(3MLIB), mlib_SignalLinear2ADPCM4Bits(3MLIB), mlib_SignalLinear2ADPCM5Bits(3MLIB), attributes(5)
mlib_SignalADPCMInit – adaptive differential pulse code modulation (ADPCM)

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalADPCMInit(void **state);

Description

The mlib_SignalADPCMInit() function creates the internal structure for the codec for
functions that perform adaptive differential pulse code modulation (ADPCM) in compliance
with the ITU (former CCITT) G.721, G.723, and G.726 specifications.

Parameters

The function takes the following arguments:

state Internal structure of the codec.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalADPCM2Bits2Linear(3MLIB), mlib_SignalADPCM3Bits2Linear(3MLIB),
mlib_SignalADPCM4Bits2Linear(3MLIB), mlib_SignalADPCM5Bits2Linear(3MLIB),
mlib_SignalADPCMFree(3MLIB), mlib_SignalLinear2ADPCM2Bits(3MLIB),
mlib_SignalLinear2ADPCM3Bits(3MLIB), mlib_SignalLinear2ADPCM4Bits(3MLIB),
mlib_SignalLinear2ADPCM5Bits(3MLIB), attributes(5)
mlib_SignalALaw2Linear(3MLIB)

Name
mlib_SignalALaw2Linear – ITU G.711 m-law and A-law compression and decompression

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalALaw2Linear(mlib_s16 *pcm, const mlib_u8 *acode,
                                   mlib_s32 n);

Description
The mlib_SignalALaw2Linear() function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

Parameters
The function takes the following arguments:

pcm Linear PCM sample array.
acode A-law code array.
n Number of samples in the source array.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalALaw2uLaw(3MLIB), mlib_SignalLinear2ALaw(3MLIB),
mlib_SignalLinear2uLaw(3MLIB), mlib_SignaluLaw2ALaw(3MLIB),
mlib_SignaluLaw2Linear(3MLIB), attributes(5)
**Name**
mlib_SignalALaw2uLaw – ITU G.711 m-law and A-law compression and decompression

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalALaw2uLaw(mlib_u8 *ucode, const mlib_u8 *acode,
                                 mlib_s32  n);
```

**Description**
The `mlib_SignalALaw2uLaw()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

**Parameters**
The function takes the following arguments:
- `ucode` m-law code array.
- `acode` A-law code array.
- `n` Number of samples in the source array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_SignalALaw2Linear(3MLIB)`, `mlib_SignalLinear2ALaw(3MLIB)`,
`mlib_SignalLinear2uLaw(3MLIB)`, `mlib_SignaluLaw2ALaw(3MLIB)`,
`mlib_SignaluLaw2Linear(3MLIB)`, attributes(5)
Name  mlib_SignalAutoCorrel_S16, mlib_SignalAutoCorrel_S16S, mlib_SignalAutoCorrel_F32, mlib_SignalAutoCorrel_F32S – signal auto-correlation

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalAutoCorrel_S16(mlib_d64 *correl,
    const mlib_s16 *src, mlib_s32 disp, mlib_s32 n);
mlib_status mlib_SignalAutoCorrel_S16S(mlib_d64 *correl,
    const mlib_s16 *src, mlib_s32 disp, mlib_s32 n);
mlib_status mlib_SignalAutoCorrel_F32(mlib_d64 *correl,
    const mlib_f32 *src, mlib_s32 disp, mlib_s32 n);
mlib_status mlib_SignalAutoCorrel_F32S(mlib_d64 *correl,
    const mlib_f32 *src, mlib_s32 disp, mlib_s32 n);

Description  Each of these functions performs auto-correlation.

For monaural signals, the following equation is used:

\[
\text{correl}[0] = \frac{1}{n-d-1} \sum_{i=0}^{n-d-1} (\text{src}[i] \times \text{src}[i + d])
\]

For stereo signals, the following equation is used:

\[
\begin{align*}
\text{correl}[0] &= \frac{1}{n-d-1} \sum_{i=0}^{n-d-1} (\text{src}[2*i] \times \text{src}[2*(i + d)]) \\
\text{correl}[1] &= \frac{1}{n-d-1} \sum_{i=0}^{n-d-1} (\text{src}[2*i + 1] \times \text{src}[2*(i + d) + 1])
\end{align*}
\]

where \(d = \text{disp}\).

Parameters  Each of the functions takes the following arguments:

- \textit{correl}: Pointer to the auto-correlation array. In the stereo version, \text{correl}[0] contains the auto-correlation of channel 0, and \text{correl}[1] contains the auto-correlation of channel 1.
- \textit{src}: Source signal array.
- \textit{disp}: Displacement. \(0 \leq \text{disp} < n\).
- \textit{n}: Number of samples in the source signal array.
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlb_SignalCrossCorrel_S16(3MLIB), attributes(5)
### Name
mlib_SignalCepstral_F32 – perform cepstral analysis

### Synopsis
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalCepstral_F32(mlib_f32 *cepst,
    const mlib_f32 *signal, void *state);
```

### Description
The `mlib_SignalCepstral_F32()` function performs cepstral analysis.

The basic operations to compute the cepstrum is shown below.

```
+-----------+ +--------+ +-----------+
| Fourier   | | log|*| | Fourier |
----->| |----->|---| |------>| Fourier |----->
-------| |-------| | -|--| |-------| |-------|
x(n) | Transform | X(k) | | X'(k) | Transform | c(n)
+-----------+ +--------+ +-----------+
```

where \(x(n)\) is the input signal and \(c(n)\) is its cepstrum. In mathematics, they are

\[
X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j\frac{2\pi kn}{N})
\]

\[
X'(k) = \log|X(k)|
\]

\[
c(n) = \sum_{n=0}^{N-1} X'(k) \exp(j\frac{2\pi kn}{N})
\]

Since \(X'(k)\) is real and even (symmetric), i.e.

\[
X'(k) = X'(N - k)
\]

the \(c(n)\) is real and the equation becomes Cosine transform.

\[
c(n) = \sum_{n=0}^{N-1} X'(k) \cos\left(\frac{2\pi kn}{N}\right)
\]

The cepstral coefficients in LPC is a special case of the above.


The function takes the following arguments:

- `cepst`  The cepstral coefficients.
- `signal`  The input signal vector.
- `state`  Pointer to the internal state structure.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  `mlib_SignalCepstralInit_F32(3MLIB)`, `mlib_SignalCepstralFree_F32(3MLIB)`, `attributes(5)`
# mlib_SignalCepstralFree_S16(3MLIB)

**Name**
mlib_SignalCepstralFree_S16, mlib_SignalCepstralFree_F32 – clean up for cepstral analysis

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalCepstralFree_S16(void *state);
void mlib_SignalCepstralFree_F32(void *state);
```

**Description**
Each of these functions frees the internal state structure for cepstral analysis.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters**
Each of the functions takes the following arguments:
- `state` Pointer to the internal state structure.

**Return Values**
None.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalCepstral_S16(3MLIB), mlib_SignalCepstral_F32(3MLIB), mlib_SignalCepstral_S16_Adp(3MLIB), mlib_SignalCepstralInit_S16(3MLIB), mlib_SignalCepstralInit_F32(3MLIB), attributes(5)
Name: mlib_SignalCepstralInit_S16, mlib_SignalCepstralInit_F32 – initialization for cepstral analysis

Synopsis:

```c
#include <mlib.h>

mlib_status mlib_SignalCepstralInit_S16(void *state, mlib_s32 order);
mlib_status mlib_SignalCepstralInit_F32(void *state, mlib_s32 order);
```

Description:

Each of these functions initializes the internal state structure for cepstral analysis.

The `init` function performs internal state structure allocation and global initialization. Per function call initialization is done in each function, so the same internal state structure can be reused for multiple function calls.

Parameters:

Each of the functions takes the following arguments:

- `order`  The order of the input signal vector and the cepstral coefficients, where *length* = \(2^{order}\).
- `state` Pointer to the internal state structure.

Return Values:

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes:

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also:

- mlib_SignalCepstral_S16(3MLIB), mlib_SignalCepstral_S16_Adp(3MLIB), mlib_SignalCepstral_F32(3MLIB), mlib_SignalCepstralFree_S16(3MLIB), mlib_SignalCepstralFree_F32(3MLIB), attributes(5)
The `mlib_SignalCepstral_S16()` function performs cepstral analysis. The user supplied scaling factor will be used and the output will be saturated if necessary.

The basic operations to compute the cepstrum is shown below.

```
+--------+ +--------+ +-----------+
| Fourier | | | | Inverse |
+--------+ +--------+ +-----------+
```

where \( x(n) \) is the input signal and \( c(n) \) is its cepstrum. In mathematics, they are

\[
X(k) = \sum_{n=0}^{N-1} x(n) \times \exp(-j \frac{2\pi k n}{N})
\]

\[
c(n) = \frac{1}{N} \sum_{k=0}^{N-1} \log|X(k)| \times \exp(j \frac{2\pi k n}{N})
\]

Since \( X'(k) \) is real and even (symmetric), i.e.

\[
X'(k) = X'(N - k)
\]

the \( c(n) \) is real and the equation becomes Cosine transform.

\[
c(n) = \frac{1}{N} \sum_{k=0}^{N-1} X'(k) \times \cos(\frac{2\pi k n}{N})
\]

The cepstral coefficients in LPC is a special case of the above.


The function takes the following arguments:

- **cepst**: The cepstral coefficients.
- **cscale**: The scaling factor of cepstral coefficients, where actual_data = output_data * 2**(-scaling_factor).
- **signal**: The input signal vector, the signal samples are in Q15 format.
- **state**: Pointer to the internal state structure.

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

See **attributes(5)** for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also **mlib_SignalCepstralInit_S16(3MLIB)**, **mlib_SignalCepstral_S16_Adpt(3MLIB)**, **mlib_SignalCepstralFree_S16(3MLIB)**, **attributes(5)**
The \texttt{mlib_SignalCepstral_S16_Adp()} function performs cepstral analysis. The scaling factor of the output data will be calculated based on the actual data.

The basic operations to compute the cepstrum is shown below.

\begin{center}
\begin{tabular}{c|c|c|c|c|c}
| Fourier | | | | Inverse | \\
\hline
\rightarrow| |\rightarrow| log|\rightarrow| Fourier |
\hline
x(n) | Transform | X(k) | | X'(k) | Transform | c(n) \\
\hline
| +---------+ | +--------+ | +-----------+ |
\end{tabular}
\end{center}

where \( x(n) \) is the input signal and \( c(n) \) is its cepstrum. In mathematics, they are

\[
X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j\frac{2\pi k n}{N})
\]

\[
X'(k) = \log|X(k)|
\]

\[
c(n) = \sum_{n=0}^{N-1} X'(k) \exp(j\frac{2\pi k n}{N})
\]

Since \( X'(k) \) is real and even (symmetric), i.e.

\[
X'(k) = X'(N - k)
\]

the \( c(n) \) is real and the equation becomes Cosine transform.

\[
c(n) = \sum_{n=0}^{N-1} X'(k) \cos\left(j\frac{2\pi k n}{N}\right)
\]

The cepstral coefficients in LPC is a special case of the above.


The function takes the following arguments:

cp

The cepstral coefficients.

cscale

The scaling factor of cepstral coefficients, where actual_data = output_data * 2**(-scaling_factor).

signal

The input signal vector, the signal samples are in Q15 format.

state

Pointer to the internal state structure.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalCepstralInit_S16(3MLIB), mlib_SignalCepstral_S16(3MLIB), mlib_SignalCepstralFree_S16(3MLIB), attributes(5)
Each of these functions performs data type convert with shifting.

The following equation is used:
\[ \text{dst}[i] = \text{src}[i] \times 2^{\text{shift}} \]

See the following table for available variations of this group of data type convert functions.

<table>
<thead>
<tr>
<th>Type[^]</th>
<th>F32</th>
<th>F32S</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>S16</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>S32</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>U8S</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
Each row represents a source data type. Each column represents a destination data type.

**Parameters**

Each of the functions takes the following arguments:

- `dst` Destination signal array.
- `src` Source signal array.
- `shift` Left shifting factor.
- `n` Number of samples in the source signal arrays.

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_SignalConvertShift_F32_U8_Sat(3MLIB)`, attributes(5)
**Name**
mlib_SignalConvertShift_U8_S8_Sat, mlib_SignalConvertShift_U8_S16_Sat,
mlib_SignalConvertShift_U8_S32_Sat, mlib_SignalConvertShift_U8_F32_Sat,
mlib_SignalConvertShift_U8S_S8S_Sat, mlib_SignalConvertShift_U8S_S16S_Sat,
mlib_SignalConvertShift_U8S_S32S_Sat, mlib_SignalConvertShift_U8S_F32S_Sat,
mlib_SignalConvertShift_S8_U8_Sat, mlib_SignalConvertShift_S8_S16_Sat,
mlib_SignalConvertShift_S8_S32_Sat, mlib_SignalConvertShift_S8_F32_Sat,
mlib_SignalConvertShift_S8S_U8S_Sat, mlib_SignalConvertShift_S8S_S16S_Sat,
mlib_SignalConvertShift_S8S_S32S_Sat, mlib_SignalConvertShift_S8S_F32S_Sat,
mlib_SignalConvertShift_S16_U8_Sat, mlib_SignalConvertShift_S16_S8_Sat,
mlib_SignalConvertShift_S16_S32_Sat, mlib_SignalConvertShift_S16_F32_Sat,
mlib_SignalConvertShift_S16S_U8S_Sat, mlib_SignalConvertShift_S16S_S8S_Sat,
mlib_SignalConvertShift_S16S_S16S_Sat, mlib_SignalConvertShift_S16S_F32S_Sat,
mlib_SignalConvertShift_S32_U8_Sat, mlib_SignalConvertShift_S32_S8_Sat,
mlib_SignalConvertShift_S32_S16_Sat, mlib_SignalConvertShift_S32_F32_Sat,
mlib_SignalConvertShift_S32S_U8S_Sat, mlib_SignalConvertShift_S32S_S8S_Sat,
mlib_SignalConvertShift_S32S_S16S_Sat, mlib_SignalConvertShift_S32S_F32S_Sat

Data type convert with shifting

**Synopsis**

```
c [ flag ... ] file ... -lmlib [ library ... ]
#include <mlib.h>

mlib_status mlib_SignalConvertShift_U8_S8_Sat(mlib_u8 *dst,
    const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8_S16_Sat(mlib_u8 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8_S32_Sat(mlib_u8 *dst,
    const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8_F32_Sat(mlib_u8 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_S8S_Sat(mlib_u8 *dst,
    const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_S16S_Sat(mlib_u8 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_S32S_Sat(mlib_u8 *dst,
    const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_U8S_F32S_Sat(mlib_u8 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8_U8_Sat(mlib_s8 *dst,
    const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8_S16_Sat(mlib_s8 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8_S32_Sat(mlib_s8 *dst,
    const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8_F32_Sat(mlib_s8 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8S_U8S_Sat(mlib_s8 *dst,
    const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8S_S16S_Sat(mlib_s8 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8S_S32S_Sat(mlib_s8 *dst,
    const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S8S_F32S_Sat(mlib_s8 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16_U8_Sat(mlib_s16 *dst,
    const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16_S8_Sat(mlib_s16 *dst,
    const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16_S32_Sat(mlib_s16 *dst,
    const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16_F32_Sat(mlib_s16 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16S_U8S_Sat(mlib_s16 *dst,
    const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16S_S8S_Sat(mlib_s16 *dst,
    const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S16S_F32S_Sat(mlib_s16 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32_U8_Sat(mlib_s32 *dst,
    const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32_S8_Sat(mlib_s32 *dst,
    const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32_S16_Sat(mlib_s32 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32_F32_Sat(mlib_s32 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32S_U8S_Sat(mlib_s32 *dst,
    const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32S_S8S_Sat(mlib_s32 *dst,
    const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32S_S16S_Sat(mlib_s32 *dst,
    const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalConvertShift_S32S_F32S_Sat(mlib_s32 *dst,
    const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
```
mlib_status mlib_SignalConvertShift_S8_S32_Sat(mlib_s8 *dst, const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S8_F32_Sat(mlib_s8 *dst, const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S8S_U8S_Sat(mlib_s8 *dst, const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S8S_S16S_Sat(mlib_s8 *dst, const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S8S_S32S_Sat(mlib_s8 *dst, const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S8S_F32S_Sat(mlib_s8 *dst, const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16_U8_Sat(mlib_s16 *dst, const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16_S8_Sat(mlib_s16 *dst, const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16_S32_Sat(mlib_s16 *dst, const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16_F32_Sat(mlib_s16 *dst, const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16S_U8S_Sat(mlib_s16 *dst, const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16S_S8S_Sat(mlib_s16 *dst, const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16S_S32S_Sat(mlib_s16 *dst, const mlib_s32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S16S_F32S_Sat(mlib_s16 *dst, const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32_U8_Sat(mlib_s32 *dst, const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32_S8_Sat(mlib_s32 *dst, const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32_S16_Sat(mlib_s32 *dst, const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32_F32_Sat(mlib_s32 *dst, const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_U8S_Sat(mlib_s32 *dst, const mlib_u8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_S8S_Sat(mlib_s32 *dst, const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_S16S_Sat(mlib_s32 *dst, const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_F32S_Sat(mlib_s32 *dst, const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_S8S_Sat(mlib_s32 *dst,
const mlib_s8 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_S16S_Sat(mlib_s32 *dst,
const mlib_s16 *src, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalConvertShift_S32S_F32S_Sat(mlib_s32 *dst,
const mlib_f32 *src, mlib_s32 shift, mlib_s32 n);

**Description**

Each of these functions performs data type convert with shifting.

The following equation is used:

\[ \text{dst}[i] = \text{saturate}(\text{src}[i] \times 2^{\text{shift}}) \]

See the following tables for available variations of this group of data type convert functions.

<table>
<thead>
<tr>
<th>Type[*]</th>
<th>U8</th>
<th>S8</th>
<th>S16</th>
<th>S32</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S8</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S16</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S32</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>F32</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type[*]</th>
<th>U8S</th>
<th>S8S</th>
<th>S16S</th>
<th>S32S</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8S</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S8S</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S16S</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S32S</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>F32S</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[*] Each row represents a source data type. Each column represents a destination data type.

**Parameters**

Each of the functions takes the following arguments:

- **dst**  Destination signal array.
- **src**  Source signal array.
- **shift**  Left shifting factor.
- **n**  Number of samples in the source signal arrays.
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalConvertShift_F32_U8(3MLIB), attributes(5)
Each of these functions performs convolution.

For monaural signals, the following equation is used:

\[ \text{dst}[i] = \sum_{j=0}^{m-1} (\text{src1}[j] \times \text{src2}[i - j]) \quad \text{if } m \leq n \]

\[ \text{dst}[i] = \sum_{j=0}^{n-1} (\text{src2}[j] \times \text{src1}[i - j]) \quad \text{if } m > n \]

where \( i = 0, 1, \ldots, (m + n - 2) \).

For stereo signals, the following equation is used:

\[ \text{dst}[2*i] = \sum_{j=0}^{m-1} (\text{src1}[2*j] \times \text{src2}[2*(i - j)]) \]

\[ \text{dst}[2*i + 1] = \sum_{j=0}^{m-1} (\text{src1}[2*j + 1] \times \text{src2}[2*(i - j) + 1]) \]

if \( m \leq n \), or

\[ \text{dst}[2*i] = \sum_{j=0}^{n-1} (\text{src2}[2*j] \times \text{src1}[2*(i - j)]) \]

\[ \text{dst}[2*i + 1] = \sum_{j=0}^{n-1} (\text{src2}[2*j + 1] \times \text{src1}[2*(i - j) + 1]) \]

if \( n > m \).
\[
\text{dst}[2i + 1] = \sum_{j=0}^{n-1} (\text{src2}[2j + 1] * \text{src1}[2(i - j) + 1])
\]

if \( m > n \); where \( i = 0, 1, \ldots, (m + n - 2) \).

**Parameters**

Each of the functions takes the following arguments:

- **\( dst \)**: Destination signal array.
- **\( src1 \)**: First source signal array.
- **\( src2 \)**: Second source signal array.
- **\( m \)**: Number of samples in the first source signal array.
- **\( n \)**: Number of samples in the second source signal arrays.

**Return Values**

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See [attributes](attributes(5)) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** [attributes](attributes(5))
mlib_SignalCrossCorrel_S16, mlib_SignalCrossCorrel_S16S, mlib_SignalCrossCorrel_F32,
mlib_SignalCrossCorrel_F32S – signal cross correlation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalCrossCorrel_S16(mlib_d64 *correl,
    const mlib_s16 *src1, const mlib_s16 *src2, mlib_s32 n);
mlib_status mlib_SignalCrossCorrel_S16S(mlib_d64 *correl,
    const mlib_s16 *src1, const mlib_s16 *src2, mlib_s32 n);
mlib_status mlib_SignalCrossCorrel_F32(mlib_d64 *correl,
    const mlib_f32 *src1, const mlib_f32 *src2, mlib_s32 n);
mlib_status mlib_SignalCrossCorrel_F32S(mlib_d64 *correl,
    const mlib_f32 *src1, const mlib_f32 *src2, mlib_s32 n);

Description

Each of these functions performs cross correlation.

For monaural signals, the following equation is used:

\[
\text{correl}[0] = \frac{1}{n} \sum_{i=0}^{n-1} (\text{src1}[i] \times \text{src2}[i])
\]

For stereo signals, the following equation is used:

\[
\text{correl}[0] = \frac{1}{n} \sum_{i=0}^{n-1} (\text{src1}[2*i] \times \text{src2}[2*i])
\]
\[
\text{correl}[1] = \frac{1}{n} \sum_{i=0}^{n-1} (\text{src1}[2*i + 1] \times \text{src2}[2*i + 1])
\]

Parameters

Each of the functions takes the following arguments:

- **correl** Pointer to the cross correlation array. In the stereo version, \(\text{correl}[0]\) contains the cross correlation of channel 0, and \(\text{correl}[1]\) contains the cross correlation of channel 1.
- **src1** First source signal array.
- **src2** Second source signal array.
- **n** Number of samples in the source signal arrays.

Return Values

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.
Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalAutoCorrel_S16(3MLIB), attributes(5)
mlib_SignalDownSample_S16_S16, mlib_SignalDownSample_S16S_S16S, mlib_SignalDownSample_F32_F32, mlib_SignalDownSample_F32S_F32S – signal downsampling

**Synopsis**

cc [ flag... ] file... -lmllib [ library... ]

```
#include <mlib.h>

mlib_status mlib_SignalDownSample_S16_S16(mlib_s16 *dst,
                                            const mlib_s16 *src,
                                            mlib_s32 factor,
                                            mlib_s32 phase,
                                            mlib_s32 n);

mlib_status mlib_SignalDownSample_S16S_S16S(mlib_s16 *dst,
                                             const mlib_s16 *src,
                                             mlib_s32 factor,
                                             mlib_s32 phase,
                                             mlib_s32 n);

mlib_status mlib_SignalDownSample_F32_F32(mlib_f32 *dst,
                                            const mlib_f32 *src,
                                            mlib_s32 factor,
                                            mlib_s32 phase,
                                            mlib_s32 n);

mlib_status mlib_SignalDownSample_F32S_F32S(mlib_f32 *dst,
                                             const mlib_f32 *src,
                                             mlib_s32 factor,
                                             mlib_s32 phase,
                                             mlib_s32 n);
```

**Description**

Each of these functions performs downsampling.

For monaural signals, the following equation is used:

\[
\text{dst}[i] = \text{src}[i \times \text{factor} + \text{phase}]
\]

where \(i = 0, 1, \ldots, (n - 1 - \text{phase})/\text{factor}\).

For stereo signals, the following equation is used:

\[
\begin{align*}
\text{dst}[2i] &= \text{src}[2(i \times \text{factor} + \text{phase})] \\
\text{dst}[2i + 1] &= \text{src}[2(i \times \text{factor} + \text{phase}) + 1]
\end{align*}
\]

where \(i = 0, 1, \ldots, (n - 1 - \text{phase})/\text{factor}\).

**Parameters**

Each of the functions takes the following arguments:

- **dst** Output signal array.
- **src** Input signal array.
- **factor** Factor by which to downsample. factor \(\geq 1\).
- **phase** Parameter that determines relative position of an output value, within the input signal. \(0 \leq \text{phase} < \text{factor}\).
- **n** Number of samples in the input signal array.
Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalUpSample_S16_S16(3MLIB), attributes(5)
mlib_SignalDTWKScalar_F32 – perform dynamic time warping for K-best paths on scalar data

Synopsis

```c
#include <mlib.h>

mlib_status mlib_SignalDTWKScalar_F32(mlib_d64 *dist,
const mlib_f32 *dobs,
mib_s32 obs,
void *state);
```

Description

The mlib_SignalDTWKScalar_F32() function performs dynamic time warping for K-best paths on scalar data.

Assume the reference data are

\( r(y), y=1,2,\ldots,N \)

and the observed data are

\( o(x), x=1,2,\ldots,M \)

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = (px(i),py(i)), i=1,2,\ldots,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)), o(px(i))) \times m(px(i),py(i))
\]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o; m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
\text{L-1} \quad d(r,o) = \sum_{i=0}^{L} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
\text{L-1} \quad d(r,o) = \text{SORT} \{ \sum_{i=0}^{L} (r(i) - o(i))^{*2} \}
\]

where \( L \) is the length of each data vector.

To scalar data where \( L=1 \), the two norms are the same.
The constraints of dynamic time warping are:

1. **Endpoint Constraints**
   
   \[ p_x(1) = 1 \]
   \[ 1 \leq p_y(1) \leq 1 + \delta \]

   and

   \[ p_x(Q) = M \]
   \[ N - \delta \leq p_y(Q) \leq N \]

2. **Monotonicity Conditions**
   
   \[ p_x(i) \leq p_x(i+1) \]
   \[ p_y(i) \leq p_y(i+1) \]

3. **Local Continuity Constraints**

   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   **Itakura Type:**

   ![Itakura Type Diagram]

   Allowable paths are:
   
   - \( p_1 \rightarrow p_0 \) \((1,0)\)
   - \( p_2 \rightarrow p_0 \) \((1,1)\)
   - \( p_3 \rightarrow p_0 \) \((1,2)\)

   Consecutive \((1,0)\) \((1,0)\) is disallowed. So path \( p_4 \rightarrow p_1 \rightarrow p_0 \) is disallowed.

4. **Global Path Constraints**

   Due to local continuity constraints, certain portions of the \((p_x, p_y)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. **Slope Weighting**

   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

A path in \((p_x, p_y)\) plane can be represented in chain code. The value of the chain code is defined as following.
### Chain Code Table

<table>
<thead>
<tr>
<th>Shift (x, y)</th>
<th>Chain Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

The numbers represent the values of the chain code for the segment that ends at the point, starting from the marked point.

In the following example, the observed data with 11 data points are mapped into the reference data with 9 data points:

```
4 | * * * * * * * * * *
1 | * * * * * * * * * *
2 | /               /
3 | * * * * * * * * * *
4 | /               /
5 | * * * * * * * * * *
6 | /               /
7 | * * * * * * * * * *
8 | /               /
9 | * * * * * * * * * *
```

where `x` marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.
The chain code that represents the path is

\( \langle 2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0 \rangle \)


**Parameters**  The function takes the following arguments:

- `dist`  The distances of the K-best paths.
- `dobs`  The observed data array.
- `lobs`  The length of the observed data array.
- `state`  Pointer to the internal state structure.

**Return Values**  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_SignalDTWKScalar_Init_F32(3MLIB), mlib_SignalDTWKScalar_F32(3MLIB), mlib_SignalDTWKScalarPath_F32(3MLIB), mlib_SignalDTWKScalarFree_F32(3MLIB), attributes(5)
**Name**  
mlib_SignalDTWKScalarFree_S16, mlib_SignalDTWKScalarFree_F32 – clean up for K-best paths of scalar data

**Synopsis**  
cc [ flag... ] file... -cmlib [ library... ]  
#include <mlib.h>  
void mlib_SignalDTWKScalarFree_S16(void *state);  
void mlib_SignalDTWKScalarFree_F32(void *state);

**Description**  
Each of these functions frees the internal state structure for dynamic time warping (DTW) for K-best paths of scalar data.

This function cleans up the internal state structure and releases all memory buffers.

**Parameters**  
Each of the functions takes the following arguments:

- state  
  Pointer to the internal state structure.

**Return Values**  
None.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalDTWKScalarInit_S16(3MLIB), mlib_SignalDTWKScalarInit_F32(3MLIB), mlib_SignalDTWKScalar_S16(3MLIB), mlib_SignalDTWKScalar_F32(3MLIB), mlib_SignalDTWKScalarPath_S16(3MLIB), mlib_SignalDTWKScalarPath_F32(3MLIB), attributes(5)
The `mlib_SignalDTWKScalarInit_F32()` function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

The function takes the following arguments:

- `dref` The reference data array.
- `lref` The length of the reference data array.
- `kbest` The number of the best paths evaluated.
- `delta` The delta in the endpoint constraints.
- `local` The type of the local continuity constraints. MLIB_DTW_ITAKURA for Itakura type constraints.
- `slope` The type of the slope weighting. MLIB_DTW_NONE for no slope weighting.
- `state` Pointer to the internal state structure.

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:
mlib_SignalDTWKScalarInit_S16 – initialization for K-best paths of scalar data

Synopsis

cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

mlib_status mlib_SignalDTWKScalarInit_S16(void **state,  
const mlib_s16 *dref, mlib_s32 lref, mlib_s32 kbest,  
mlib_s32 sref, mlib_s32 delta, mlib_s32 local,  
mlib_s32 slope);

Description

The mlib_SignalDTWKScalarInit_S16() function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

Parameters

The function takes the following arguments:

- dref: The reference data array.
- lref: The length of the reference data array.
- kbest: The number of the best paths evaluated.
- sref: The scaling factor of the reference data array, where actual_data = input_data * 2**(-scaling_factor).
- delta: The delta in the endpoint constraints.
- local: The type of the local continuity constraints. MLIB_DTW_ITAKURA for Itakura type constraints.
- slope: The type of the slope weighting. MLIB_DTW_NONE for no slope weighting.
- state: Pointer to the internal state structure.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalDTWKScalarInit_S16(3MLIB), mlib_SignalDTWKScalar_S16(3MLIB),  
mlib_SignalDTWKScalarPath_S16(3MLIB), mlib_SignalDTWKScalarFree_S16(3MLIB),  
attributes(5)
Name  mlib_SignalDTWKScalarPath_S16, mlib_SignalDTWKScalarPath_F32 – return K-best path on scalar data

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWKScalarPath_S16(mlib_s32 *path,
                              mlib_s32 *lpath, mlib_s32 kpath, void *state);

mlib_status mlib_SignalDTWKScalarPath_F32(mlib_s32 *path,
                              mlib_s32 *lpath, mlib_s32 kpath, void *state);

Description  Each of these functions returns K-best path on scalar data.

Assume the reference data are
r(y), y=1,2,...,N
and the observed data are
o(x), x=1,2,...,M
the dynamic time warping is to find a mapping function (a path)
p(i) = {px(i),py(i)}, i=1,2,...,Q
with the minimum distance.
In K-best paths case, K paths with the K minimum distances are searched.
The distance of a path is defined as
\[ \text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) * m(px(i),py(i)) \]
where \( d(r,o) \) is the dissimilarity between data point/vector r and data point/vector o;
m(x,y) is the path weighting coefficient associated with path point \((x,y)\); N is the length of the reference data; M is the length of the observed data; Q is the length of the path.

Using L1 norm (sum of absolute differences)
\[ d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)| \]

Using L2 norm (Euclidean distance)
\[ d(r,o) = \sqrt{\sum_{i=0}^{L-1} (r(i) - o(i))^2} \]
where L is the length of each data vector.
To scalar data where \( L = 1 \), the two norms are the same.

\[
d(r, o) = |r - o| = \sqrt{(r - o)^2}
\]

The constraints of dynamic time warping are:

1. Endpoint constraints
   \[
   \begin{align*}
   px(1) &= 1 \\
   1 &\leq py(1) \leq 1 + \text{delta}
   \end{align*}
   \]
   and
   \[
   \begin{align*}
   px(Q) &= M \\
   N - \text{delta} &\leq py(Q) \leq N
   \end{align*}
   \]

2. Monotonicity Conditions
   \[
   \begin{align*}
   px(i) &\leq px(i+1) \\
   py(i) &\leq py(i+1)
   \end{align*}
   \]

3. Local Continuity Constraints
   
   See Table 4.5 on page 211 in Rabiner and Juang's book.

   **Itakura Type:**
   
   \[
   \begin{array}{c}
   \text{py} \\
   \text{*----*----*} \\
   | p4 | p1 | p0 \\
   | | | \\
   *----*----* \\
   | | p2 \\
   | | | \\
   *----*----*-- px
   \end{array}
   \]

   Allowable paths are
   
   - \( p1 \rightarrow p0 \) \( (1, 0) \)
   - \( p2 \rightarrow p0 \) \( (1, 1) \)
   - \( p3 \rightarrow p0 \) \( (1, 2) \)

   Consecutive \( (1, 0) \) \( (1, 0) \) is disallowed. So path \( p4 \rightarrow p1 \rightarrow p0 \) is disallowed.

4. Global Path Constraints
   
   Due to local continuity constraints, certain portions of the \( (px, py) \) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting
   
   See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in \( (px, py) \) plane can be represented in chain code. The value of the chain code is defined as following.
<table>
<thead>
<tr>
<th>shift (x, y)</th>
<th>chain code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

where \( x \) marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
* 8 7 *
|
* 4 * 6
|
1 2 3 5
|
x--0--*--*-- px
```

```
The chain code that represents the path is

\[(2 \ 2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


**Parameters**

Each of the functions takes the following arguments:

- **path** The optimal path.
- **lpath** The length of the optimal path.
- **kpath** The path index, \(0 \leq kpath < kbest\).
- **state** Pointer to the internal state structure.

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- mlib_SignalDTWKScalarInit_S16(3MLIB), mlib_SignalDTWKScalarInit_F32(3MLIB),
- mlib_SignalDTWKScalar_S16(3MLIB), mlib_SignalDTWKScalar_F32(3MLIB),
- mlib_SignalDTWKScalarFree_S16(3MLIB), mlib_SignalDTWKScalarFree_F32(3MLIB),
- attributes(5)
Name  mlib_SignalDTWKScalar_S16 – perform dynamic time warping for K-best paths on scalar data

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWKScalar_S16(mlib_d64 *dist,
       const mlib_s16 *dobs, mlib_s32 lobs, mlib_s32 sobs,
       void *state);

Description  The mlib_SignalDTWKScalar_S16() function performs dynamic time warping for K-best paths on scalar data.

Assume the reference data are

\[ r(y), \ y=1,2,...,N \]

and the observed data are

\[ o(x), \ x=1,2,...,M \]

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = \{px(i),py(i)\}, \ i=1,2,...,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) \cdot m(px(i),py(i))
\]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
L-1
\quad d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
L-1
\quad d(r,o) = \sqrt{\sum_{i=0}^{L-1} (r(i) - o(i))^2}
\]

where \( L \) is the length of each data vector.
To scalar data where L=1, the two norms are the same.

\[ d(r,o) = |r - o| = \sqrt{(r - o)^2} \]

The constraints of dynamic time warping are:

1. Endpoint constraints

   \[ px(1) = 1 \]
   \[ 1 \leq py(1) \leq 1 + \delta \]
   and
   \[ px(Q) = M \]
   \[ N - \delta \leq py(Q) \leq N \]

2. Monotonicity Conditions

   \[ px(i) \leq px(i+1) \]
   \[ py(i) \leq py(i+1) \]

3. Local Continuity Constraints

   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   Itakura Type:

   | *----*----* |
   | | p4 | p1 | p0 |
   | *----*----* |
   | | | p2 |
   | *----*----*-- px |
   | p3 |

   Allowable paths are

   p1->p0 (1,0)  
   p2->p0 (1,1)  
   p3->p0 (1,2)  

   Consecutive (1,0) (1,0) is disallowed. So path p4->p1->p0 is disallowed.

4. Global Path Constraints

   Due to local continuity constraints, certain portions of the \((px, py)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

A path in \((px, py)\) plane can be represented in chain code. The value of the chain code is defined as following.
<table>
<thead>
<tr>
<th>shift (x, y)</th>
<th>chain code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points.

```
py
<table>
<thead>
<tr>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1235</td>
</tr>
<tr>
<td>x-0--<em>--</em>-- px</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>123</td>
</tr>
</tbody>
</table>
```

man pages section 3: Multimedia Library Functions • Last Revised 23 May 2007
The chain code that represents the path is

(2 2 2 1 2 0 2 2 0 2 0)


**Parameters** The function takes the following arguments:

- `dist` The distances of the K-best paths.
- `dobs` The observed data array.
- `lobs` The length of the observed data array.
- `sobs` The scaling factor of the observed data array, where `actual_data = input_data * 2**(-scaling_factor)`.  
- `state` Pointer to the internal state structure.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalDTWKScalarInit_S16(3MLIB), mlib_SignalDTWKScalar_S16(3MLIB), mlib_SignalDTWKScalarPath_S16(3MLIB), mlib_SignalDTWKScalarFree_S16(3MLIB), attributes(5)
The `mlib_SignalDTWKVector_F32()` function performs dynamic time warping for K-best paths on vector data.

Assume the reference data are

\[ r(y), \ y=1,2,...,N \]

and the observed data are

\[ o(x), \ x=1,2,...,M \]

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = (px(i),py(i)), \ i=1,2,...,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) \ast m(px(i),py(i))
\]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
L-1 \\
\text{d}(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
L-1 \\
\text{d}(r,o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 }
\]

where \( L \) is the length of each data vector.

To scalar data where \( L=1 \), the two norms are the same.
The constraints of dynamic time warping are:

1. Endpoint constraints
   
   \[
   px(1) = 1 \\
   1 \leq py(1) \leq 1 + \text{delta}
   \]
   
   and
   
   \[
   px(Q) = M \\
   N - \text{delta} \leq py(Q) \leq N
   \]

2. Monotonicity Conditions
   
   \[
   px(i) \leq px(i+1) \\
   py(i) \leq py(i+1)
   \]

3. Local Continuity Constraints
   
   See Table 4.5 on page 211 in Rabiner and Juang's book.

   Itakura Type:
   
   \[
   py
   |______________________|
   |p4|p1|p0|
   |   |   |
   |___________|
   |p2|
   |   |
   |___________|
   p3
   px
   
   Allowable paths are
   
   p1 -> p0  \quad (1, 0)
   p2 -> p0  \quad (1, 1)
   p3 -> p0  \quad (1, 2)

   Consecutive (1, 0) (1, 0) is disallowed. So path p4 -> p1 -> p0 is disallowed.

4. Global Path Constraints
   
   Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting
   
   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.
shift \((x, y)\) | chain code
-----------------------------
\[(1, 0)\] | 0
\[(0, 1)\] | 1
\[(1, 1)\] | 2
\[(2, 1)\] | 3
\[(1, 2)\] | 4
\[(3, 1)\] | 5
\[(3, 2)\] | 6
\[(1, 3)\] | 7
\[(2, 3)\] | 8

where \(x\) marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
| *
| 8 7 *
| *
| 4 6 *
| 1 2 3 5 *
| x--0--*--*-- px
```
The chain code that represents the path is

\[(2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


**Parameters**
The function takes the following arguments:
- \(\text{dist}\) The distances of the K-best paths.
- \(\text{dobs}\) The observed data array.
- \(\text{lobs}\) The length of the observed data array.
- \(\text{state}\) Pointer to the internal state structure.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalDTWKVectorInit_F32(3MLIB), mlib_SignalDTWKVector_F32(3MLIB),
mlib_SignalDTWKVectorPath_F32(3MLIB), mlib_SignalDTWKVectorFree_F32(3MLIB),
attributes(5)
mlib_SignalDTWKVectorFree_S16, mlib_SignalDTWKVectorFree_F32 – cleanup for K-best paths of vector data

#include <mlib.h>

void mlib_SignalDTWKVectorFree_S16(void *state);
void mlib_SignalDTWKVectorFree_F32(void *state);

Each of these functions frees the internal state structure for dynamic time warping (DTW) for K-best paths of vector data.

This function cleans up the internal state structure and releases all memory buffers.

Parameters

Each of the functions takes the following arguments:

state Pointer to the internal state structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalDTWKVectorFree_S16(3MLIB), mlib_SignalDTWKVectorFree_F32(3MLIB), mlib_SignalDTWKVectorInit_S16(3MLIB), mlib_SignalDTWKVectorInit_F32(3MLIB), mlib_SignalDTWKVectorPath_S16(3MLIB), mlib_SignalDTWKVectorPath_F32(3MLIB), attributes(5)
Name        mlib_SignalDTWKVectorInit_F32 – initialization for K-best paths of vector data

Synopsis     

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWKVectorInit_F32(void *state,
                                        const mlib_f32 **dref, mlib_s32 lref,
                                        mlib_s32 ndata,
                                        mlib_s32 kbest, mlib_s32 dtype,
                                        mlib_s32 delta,
                                        mlib_s32 local, mlib_s32 slope);

Description  The mlib_SignalDTWKVectorInit_F32() function initializes the internal state structure for
dynamic time warping (DTW) for K-best paths of vector data.

The init function performs internal state structure allocation and global initialization. Per
DTW function call initialization is done in DTW function, so the same internal state structure
can be reused for multiple DTW function calls.

Parameters   The function takes the following arguments:

dref        The reference data array.
lref        The length of the reference data array.
ndata        The length of each data vector.
kbest        The number of the best paths evaluated.
dtype        The type of distance metric between data vectors. MLIB_DTW_L1NORM for L1 norm of
difference (sum of absolute difference). MLIB_DTW_L2NORM for L2 norm of difference
(Euclidean distance).
delta        The delta in the endpoint constraints.
local        The type of the local continuity constraints. MLIB_DTW_ITAKURA for Itakura type
constraints.
slope        The type of the slope weighting. MLIB_DTW_NONE for no slope weighting.
state        Pointer to the internal state structure.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes    See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_SignalDTWKVectorInit_F32(3MLIB), mlib_SignalDTWKVector_F32(3MLIB),
mlib_SignalDTWKVectorPath_F32(3MLIB), mlib_SignalDTWKVectorFree_F32(3MLIB),
attributes(5)
mlib_SignalDTWKVectorInit_S16 – initialization for K-best paths of vector data

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_SignalDTWKVectorInit_S16(void *state,
const mlib_s16 **dref, mlib_s32 lref, mlib_s32 ndata,
mlib_s32 kbest, mlib_s32 dtype, mlib_s32 sref,
mlib_s32 delta, mlib_s32 local, mlib_s32 slope);

Description

The mlib_SignalDTWKVectorInit_S16() function initializes the internal state structure for dynamic time warping (DTW) for K-best paths of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

Parameters

The function takes the following arguments:

dref The reference data array.
lref The length of the reference data array.
ndata The length of each data vector.
kbest The number of the best paths evaluated.
dtype The type of distance metric between data vectors. MLIB_DTW_L1NORM for L1 norm of difference (sum of absolute difference). MLIB_DTW_L2NORM for L2 norm of difference (Euclidean distance).
sref The scaling factor of the reference data array, where actual_data = input_data * 2**(-scaling_factor).
delta The delta in the endpoint constraints.
local The type of the local continuity constraints. MLIB_DTW_ITAKURA for Itakura type constraints.
slope The type of the slope weighting. MLIB_DTW_NONE for no slope weighting.
state Pointer to the internal state structure.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
mlib_SignalDTWVectorInit_S16(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also [mlib_SignalDTWVectorInit_S16(3MLIB)], [mlib_SignalDTWVector_S16(3MLIB)], [mlib_SignalDTWVectorPath_S16(3MLIB)], [mlib_SignalDTWVectorFree_S16(3MLIB)], [attributes(5)]
**Name** mlib_SignalDTWKVectorPath_S16, mlib_SignalDTWKVectorPath_F32 – return K-best path on vector data

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWKVectorPath_S16(mlib_s32 *path,
                                      mlib_s32 *lpath, mlib_s32 kpath, void *state);
mlib_status mlib_SignalDTWKVectorPath_F32(mlib_s32 *path,
                                      mlib_s32 *lpath, mlib_s32 kpath, void *state);
```

**Description**

Each of these functions returns K-best path on vector data.

Assume the reference data are

\[ r(y), y=1,2,\ldots,N \]

and the observed data are

\[ o(x), x=1,2,\ldots,M \]

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = \{px(i),py(i)\}, i=1,2,\ldots,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) \cdot m(px(i),py(i))
\]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
\text{L-1} d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
\text{L-1} d(r,o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } \]

where \( L \) is the length of each data vector.
To scalar data where \( L=1 \), the two norms are the same.

\[
    d(r, o) = |r - o| = \sqrt{(r - o)^2}
\]

The constraints of dynamic time warping are:

1. **Endpoint constraints**
   
   \[
   p_x(1) = 1 \\
   1 \leq p_y(1) \leq 1 + \delta
   \]
   
   and
   
   \[
   p_x(Q) = M \\
   N - \delta \leq p_y(Q) \leq N
   \]

2. **Monotonicity Conditions**
   
   \[
   p_x(i) \leq p_x(i+1) \\
   p_y(i) \leq p_y(i+1)
   \]

3. **Local Continuity Constraints**
   
   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   **Itakura Type:**
   
   \[
   \begin{array}{c|c|c|}
   p_y & 1 & 1 \\
   \hline
   * & p_4 & p_1 & p_0 \\
   * & p_2 & & \\
   * & * & p_3 & \end{array}
   \]

   Allowable paths are
   
   \[
   p_1 \rightarrow p_0 \quad (1, 0) \\
   p_2 \rightarrow p_0 \quad (1, 1) \\
   p_3 \rightarrow p_0 \quad (1, 2)
   \]

   Consecutive \((1, 0) \rightarrow (1, 0)\) is disallowed. So path \( p_4 \rightarrow p_1 \rightarrow p_0 \) is disallowed.

4. **Global Path Constraints**
   
   Due to local continuity constraints, certain portions of the \((p_x, p_y)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. **Slope Weighting**
   
   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

   A path in \((p_x, p_y)\) plane can be represented in chain code. The value of the chain code is defined as following.
where \( x \) marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
|
* 8 7 *
|
* 4 6 *
|
1 2 3 5
|
x--0--*--*-- px
```
The chain code that represents the path is 
\[(2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


Parameters
Each of the functions takes the following arguments:
- `path` The optimal path.
- `lpath` The length of the optimal path.
- `kpath` The path index, \(0 \leq kpath < k\text{best}\).
- `state` Pointer to the internal state structure.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
- mlib_SignalDTWKScalarInit_S16(3MLIB)
- mlib_SignalDTWKVectorInit_F32(3MLIB)
- mlib_SignalDTWKScalar_S16(3MLIB)
- mlib_SignalDTWKVector_F32(3MLIB)
- mlib_SignalDTWKScalarFree_S16(3MLIB)
- mlib_SignalDTWKScalarFree_F32(3MLIB)
- attributes(5)
The `mlib_SignalDTWKVector_S16()` function performs dynamic time warping for K-best paths on vector data.

Assume the reference data are
\[ r(y), y=1,2,...,N \]

and the observed data are
\[ o(x), x=1,2,...,M \]

the dynamic time warping is to find a mapping function (a path)
\[ p(i) = \{px(i),py(i)\}, i=1,2,...,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as
\[ \text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) * m(px(i),py(i)) \]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)
\[ L-1 \]
\[ d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)| \]

Using L2 norm (Euclidean distance)
\[ L-1 \]
\[ d(r,o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } \]

where \( L \) is the length of each data vector.
To scalar data where $L=1$, the two norms are the same.

$$d(r, o) = |r - o| = \sqrt{(r - o)^2}$$

The constraints of dynamic time warping are:

1. **Endpoint constraints**
   
   $p_x(1) = 1$
   
   $1 \leq p_y(1) \leq 1 + \delta$

   and

   $p_x(Q) = M$
   
   $N - \delta \leq p_y(Q) \leq N$

2. **Monotonicity Conditions**
   
   $p_x(i) \leq p_x(i + 1)$
   
   $p_y(i) \leq p_y(i + 1)$

3. **Local Continuity Constraints**

   See Table 4.5 on page 211 in Rabiner and Juang's book.

   **Itakura Type:**

   $\begin{array}{ccc}
   | & | & |
   \hline
   * & * & *
   \hline
   \hline
   | & | & |
   \hline
   p4 & p1 & p0
   \end{array}$

   Allowable paths are

   $p1 \rightarrow p0 \ (1, 0)$
   
   $p2 \rightarrow p0 \ (1, 1)$
   
   $p3 \rightarrow p0 \ (1, 2)$

   Consecutive $(1, 0) (1, 0)$ is disallowed. So path $p4 \rightarrow p1 \rightarrow p0$ is disallowed.

4. **Global Path Constraints**

   Due to local continuity constraints, certain portions of the $(p_x, p_y)$ plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. **Slope Weighting**

   See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

   A path in $(p_x, p_y)$ plane can be represented in chain code. The value of the chain code is defined as following.
### Table: Chain Code Values

<table>
<thead>
<tr>
<th>shift (x, y)</th>
<th>chain code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

---

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py
| 9 | * * * * * * * * * * * | 1 |
|   | /                     |
|   | * * * * * * * * * *   |
|   | /                     |
|   | * * * * * * * * * *   |
|   | /                     |
|   | * * * * * * * * * *   |
|   | /                     |
|   | * * * * * * * * * *   |
|   | /                     |
|   | * * * * * * * * * *   |
|   | /                     |
|   | * * * * * * * * * *   |
|   | /                     |
|   | * * * * * * * * * *   |
```
The chain code that represents the path is
\[(2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


**Parameters**
The function takes the following arguments:
- *dist*  
The distances of the K-best paths.
- *dobs*  
The observed data array.
- *lobs*  
The length of the observed data array.
- *sobs*  
The scaling factor of the observed data array, where actual_data = input_data * 2**(-scaling_factor).
- *state*  
Pointer to the internal state structure.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_SignalDTWKVectorInit_S16(3MLIB), mlib_SignalDTWKVector_S16(3MLIB), mlib_SignalDTWKVectorPath_S16(3MLIB), mlib_SignalDTWKVectorFree_S16(3MLIB), attributes(5)`
The `mlib_SignalDTWScalar_F32()` function performs dynamic time warping on scalar data.

Assume the reference data are

\[ r(y), \quad y=1,2,\ldots,N \]

and the observed data are

\[ o(x), \quad x=1,2,\ldots,M \]

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = \{px(i),py(i)\}, \quad i=1,2,\ldots,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) \cdot m(px(i),py(i))
\]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
L-1 \quad d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
L-1 \quad d(r,o) = \sqrt{\sum_{i=0}^{L-1} (r(i) - o(i))^2}
\]

where \( L \) is the length of each data vector.

To scalar data where \( L=1 \), the two norms are the same.

\[
d(r,o) = |r - o| = \sqrt{(r - o)^2}
\]
The constraints of dynamic time warping are:

1. Endpoint constraints
   \[
   px(1) = 1 \\
   1 \leq py(1) \leq 1 + \text{delta}
   \]
   and
   \[
   px(Q) = M \\
   N - \text{delta} \leq py(Q) \leq N
   \]

2. Monotonicity Conditions
   \[
   px(i) \leq px(i+1) \\
   py(i) \leq py(i+1)
   \]

3. Local Continuity Constraints
   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   Itakura Type:
   
   \[
   \begin{array}{c}
   py \\
   | \cdots|\cdots| \cdots| \\
   | \ p4 \ | p1 \ | p0 \ \\
   | \ | \ | \ |
   | \cdots|\cdots| \cdots| \\
   | \ | p2 \ |
   | \ | \ | \ |
   | \cdots|\cdots|\cdots| px \\
   p3
   \end{array}
   \]

   Allowable paths are
   
   \[
   p1 \rightarrow p0 \quad (1,0) \\
   p2 \rightarrow p0 \quad (1,1) \\
   p3 \rightarrow p0 \quad (1,2)
   \]

   Consecutive (1, 0) (1, 0) is disallowed. So path p4 -> p1 -> p0 is disallowed.

4. Global Path Constraints
   Due to local continuity constraints, certain portions of the \((px, py)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting
   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

A path in \((px, py)\) plane can be represented in chain code. The value of the chain code is defined as following.

\[
\begin{array}{llllll}
\text{shift} \ (x, \ y) & \text{chain code}
\end{array}
\]

\begin{table}
\end{table}
where \( x \) marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
py
|  \\
| * 8 7 *  \\
| * 4 * 6  \\
| 1 2 3 5  \\
| x--0--*--*-- px
\end{array}
\]
The chain code that represents the path is

\( \langle 2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0 \rangle \)


**Parameters**  The function takes the following arguments:

- `dist` The distance of the optimal path.
- `dobs` The observed data array.
- `lobs` The length of the observed data array.
- `state` Pointer to the internal state structure.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_SignalDTWScalarInit_F32(3MLIB)`, `mlib_SignalDTWScalar_F32(3MLIB)`, `mlib_SignalDTWScalarPath_F32(3MLIB)`, `mlib_SignalDTWScalarFree_F32(3MLIB)`, `attributes(5)`
mlib_SignalDTWScalarFree_S16, mlib_SignalDTWScalarFree_F32 – clean up for scalar data

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalDTWScalarFree_S16(void *state);
void mlib_SignalDTWScalarFree_F32(void *state);

Description

Each of these functions frees the internal state structure for dynamic time warping (DTW) of scalar data.

This function cleans up the internal state structure and releases all memory buffers.

Parameters

Each of the functions takes the following arguments:

\textit{state} \quad \text{Pointer to the internal state structure.}

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalDTWScalarInit_S16(3MLIB), mlib_SignalDTWScalarInit_F32(3MLIB),
mlib_SignalDTWScalar_S16(3MLIB), mlib_SignalDTWScalar_F32(3MLIB),
mlib_SignalDTWScalarPath_S16(3MLIB), mlib_SignalDTWScalarPath_F32(3MLIB),
attributes(5)
### mlib_SignalDTWScalarInit_F32

- **Name**: mlib_SignalDTWScalarInit_F32 – initialization for scalar data
- **Synopsis**
  ```
  cc [ flag... ] file... -lm [ library... ]
  #include <mlib.h>
  
  mlib_status mlib_SignalDTWScalarInit_F32(void *state,
     const mlib_f32 *dref, mlib_s32 lref, mlib_s32 delta,
     mlib_s32 local, mlib_s32 slope);
  ```
- **Description**
The mlib_SignalDTWScalarInit_F32() function initializes the internal state structure for dynamic time warping (DTW) of scalar data.

  The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

- **Parameters**
The function takes the following arguments:
  - *dref*  The reference data array.
  - *lref*  The length of the reference data array.
  - *delta*  The delta in the endpoint constraints.
  - *local*  The type of the local continuity constraints. MLIB_DTW_ITAKURA for Itakura type constraints.
  - *slope*  The type of the slope weighting. MLIB_DTW_NONE for no slope weighting.
  - *state*  Pointer to the internal state structure.

- **Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

- **Attributes**
  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

- **See Also**
  mlib_SignalDTWScalarInit_F32(3MLIB), mlib_SignalDTWScalar_F32(3MLIB), mlib_SignalDTWScalarPath_F32(3MLIB), mlib_SignalDTWScalarFree_F32(3MLIB), attributes(5)
# mlib_SignalDTWScalarInit_S16

**Name**
mlib_SignalDTWScalarInit_S16 – initialization for scalar data

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWScalarInit_S16(void *state, const mlib_s16 *dref, mlib_s32 lref, mlib_s32 sref, mlib_s32 delta, mlib_s32 local, mlib_s32 slope);
```

**Description**
The `mlib_SignalDTWScalarInit_S16()` function initializes the internal state structure for dynamic time warping (DTW) of scalar data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

**Parameters**
The function takes the following arguments:
- `dref` The reference data array.
- `lref` The length of the reference data array.
- `sref` The scaling factor of the reference data array, where `actual_data = input_data * 2**(-scaling_factor)`.
- `delta` The delta in the endpoint constraints.
- `local` The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.
- `slope` The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.
- `state` Pointer to the internal state structure.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_SignalDTWScalarInit_S16(3MLIB)`
- `mlib_SignalDTWScalar_S16(3MLIB)`
- `mlib_SignalDTWScalarPath_S16(3MLIB)`
- `mlib_SignalDTWScalarFree_S16(3MLIB)`
- attributes(5)
mlib_SignalDTWScalarPath_F32 – perform dynamic time warping on scalar data

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWScalarPath_F32(mlib_d64 *dist, mlib_s32 *path,
mlib_s32 *lpath, const mlib_f32 *dobs, mlib_s32lobs, void *state);

Description

The mlib_SignalDTWScalarPath_F32() function performs dynamic time warping on scalar data.

Assume the reference data are
\[ r(y), \quad y=1,2,\ldots,N \]
and the observed data are
\[ o(x), \quad x=1,2,\ldots,M \]
the dynamic time warping is to find a mapping function (a path)
\[ p(i) = \{ px(i), py(i) \}, \quad i=1,2,\ldots,Q \]
with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as
\[
dist = \sum_{i=1}^{Q} d(r(py(i)), o(px(i))) \cdot m(px(i), py(i))
\]
where \( d(r, o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x, y) \) is the path weighting coefficient associated with path point \( (x, y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)
\[
L-1 \quad d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]
Using L2 norm (Euclidean distance)
\[
L-1 \quad d(r, o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } \]
where \( L \) is the length of each data vector.
To scalar data where \( L=1 \), the two norms are the same.
The constraints of dynamic time warping are:

1. **Endpoint constraints**
   - \( px(1) = 1 \)
   - \( 1 \leq py(1) \leq 1 + \delta \)

   and

   - \( px(Q) = M \)
   - \( N - \delta \leq py(Q) \leq N \)

2. **Monotonicity Conditions**
   - \( px(i) \leq px(i+1) \)
   - \( py(i) \leq py(i+1) \)

3. **Local Continuity Constraints**

   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   **Itakura Type:**
   
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>p4</td>
<td>p1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p0</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td></td>
<td>px</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p3</td>
</tr>
</tbody>
</table>

   Allowable paths are
   
   - \( p1 \rightarrow p0 \) (1,0)
   - \( p2 \rightarrow p0 \) (1,1)
   - \( p3 \rightarrow p0 \) (1,2)

   Consecutive (1, 0) (1, 0) is disallowed. So path \( p4 \rightarrow p1 \rightarrow p0 \) is disallowed.

4. **Global Path Constraints**

   Due to local continuity constraints, certain portions of the \((px, py)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. **Slope Weighting**

   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

   A path in \((px, py)\) plane can be represented in chain code. The value of the chain code is defined as following.
<table>
<thead>
<tr>
<th>shift (x, y)</th>
<th>chain code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```plaintext
py
| * 8 7 * |
| * 4 * 6 |
| 1 2 3 5 |
| x--0--*--*-- px
```
The chain code that represents the path is

\[(2, 2, 2, 1, 2, 0, 2, 0, 2, 0)\]


**Parameters**
The function takes the following arguments:

- `dist`  The distance of the optimal path.
- `path`  The optimal path.
- `lpath` The length of the optimal path.
- `dobs`  The observed data array.
- `lobs`  The length of the observed data array.
- `state` Pointer to the internal state structure.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalDTWScalarInit_F32(3MLIB), mlib_SignalDTWScalar_F32(3MLIB),
mlib_SignalDTWScalarPath_F32(3MLIB), mlib_SignalDTWScalarFree_F32(3MLIB),
attributes(5)
mlib_SignalDTWScalarPath_S16 – perform dynamic time warping on scalar data

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWScalarPath_S16(mlib_d64 *dist, mlib_s32 *path,
mlib_s32 *lpath, const mlib_s16 *dobs, mlib_s32 lobs, mlib_s32 sobs,
void *state);
```

Description

The `mlib_SignalDTWScalarPath_S16()` function performs dynamic time warping on scalar data.

Assume the reference data are

\[ r(y), \quad y = 1, 2, \ldots, N \]

and the observed data are

\[ o(x), \quad x = 1, 2, \ldots, M \]

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = \{px(i), py(i)\}, \quad i = 1, 2, \ldots, Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)), o(px(i))) \times m(px(i), py(i))
\]

where \( d(r, o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x, y) \) is the path weighting coefficient associated with path point \( (x, y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
L-1 \quad d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
L-1 \quad d(r, o) = \sqrt{\sum_{i=0}^{L-1} (r(i) - o(i))^2}
\]

where \( L \) is the length of each data vector.

To scalar data where \( L=1 \), the two norms are the same.
\[ d(r,o) = \|r - o\| = \sqrt{(r - o)^2} \]

The constraints of dynamic time warping are:

1. **Endpoint constraints**
   
   \[ px(1) = 1 \]
   
   \[ 1 \leq py(1) \leq 1 + \delta \]

   and

   \[ px(Q) = M \]
   
   \[ N - \delta \leq py(Q) \leq N \]

2. **Monotonicity Conditions**

   \[ px(i) \leq px(i+1) \]
   
   \[ py(i) \leq py(i+1) \]

3. **Local Continuity Constraints**

   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   **Itakura Type:**

   \[
   \begin{array}{c|c|c|c}
   \hline
   | & | & | \\
   | & | & | \\
   | & | & | \\
   p1 & p0 & p0 \\
   \hline
   \end{array}
   \]

   Allowable paths are

   \[ p1 \rightarrow p0 \quad (1,0) \]
   
   \[ p2 \rightarrow p0 \quad (1,1) \]
   
   \[ p3 \rightarrow p0 \quad (1,2) \]

   Consecutive \((1,0)\) \((1,0)\) is disallowed. So path \(p4 \rightarrow p1 \rightarrow p0\) is disallowed.

4. **Global Path Constraints**

   Due to local continuity constraints, certain portions of the \((px, py)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. **Slope Weighting**

   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

   A path in \((px, py)\) plane can be represented in chain code. The value of the chain code is defined as following.
shift (x, y) | chain code
--------------
( 1, 0 )  | 0
( 0, 1 )  | 1
( 1, 1 )  | 2
( 2, 1 )  | 3
( 1, 2 )  | 4
( 3, 1 )  | 5
( 3, 2 )  | 6
( 1, 3 )  | 7
( 2, 3 )  | 8

---
py

| 8 7 |
| 4 6 |
| 1 2 3 5 |
x---0---*---*--- px

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

py

| 9 |
| * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

py

| 9 |
| * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |
|  * * * * * * * * * * |

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.
The chain code that represents the path is

\[(2 2 2 1 2 0 2 2 0 2 0)\]


**Parameters**
The function takes the following arguments:

- `dist` The distance of the optimal path.
- `path` The optimal path.
- `lpath` The length of the optimal path.
- `dobs` The observed data array.
- `lobs` The length of the observed data array.
- `sobs` The scaling factor of the observed data array, where `actual_data = input_data * 2 ** (-scaling_factor)`.
- `state` Pointer to the internal state structure.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_SignalDTWScalarInit_S16(3MLIB)`, `mlib_SignalDTWScalar_S16(3MLIB)`, `mlib_SignalDTWScalarPath_S16(3MLIB)`, `mlib_SignalDTWScalarFree_S16(3MLIB)`, attributes(5)
# mlib_SignalDTWScalar_S16

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWScalar_S16(mlib_d64 *dist, 
const mlib_s16 *dobs, mlib_s32 lobs, mlib_s32 sobs, 
void *state);
```

**Description**

The `mlib_SignalDTWScalar_S16()` function performs dynamic time warping on scalar data.

Assume the reference data are

\[ r(y), \text{ } y=1,2,\ldots,N \]

and the observed data are

\[ o(x), \text{ } x=1,2,\ldots,M \]

the dynamic time warping is to find a mapping function (a path)

\[ p(i) = \{ px(i), py(i) \}, \text{ } i=1,2,\ldots,Q \]

with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as

\[
Q \quad \text{dist} = \sum_{i=1}^{Q} d(\text{py}(i), \text{o(px}(i))) \times \text{m(px}(i), \text{py}(i))
\]

where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o; m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)

\[
L-1 \quad d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)

\[
L-1 \quad d(r,o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } 
\]

where \( L \) is the length of each data vector.

To scalar data where \( L=1 \), the two norms are the same.
The constraints of dynamic time warping are:

1. Endpoint constraints
   \[ p_x(1) = 1 \]
   \[ 1 \leq p_y(1) \leq 1 + \delta \]
   and
   \[ p_x(Q) = M \]
   \[ N - \delta \leq p_y(Q) \leq N \]

2. Monotonicity Conditions
   \[ p_x(i) \leq p_x(i+1) \]
   \[ p_y(i) \leq p_y(i+1) \]

3. Local Continuity Constraints
   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   Itakura Type:
   \[ \begin{array}{c}
   p_y \\
   | \\
   *-----*-----* \\
   | p_4 | p_1 | p_0 \\
   | | | \\
   *-----*-----* \\
   | | p_2 | \\
   | | | \\
   *-----*-----*-- p_x \\
   p_3
   \end{array} \]

   Allowable paths are
   \[ p_1 \rightarrow p_0 \quad (1, 0) \]
   \[ p_2 \rightarrow p_0 \quad (1, 1) \]
   \[ p_3 \rightarrow p_0 \quad (1, 2) \]

   Consecutive \((1, 0) (1, 0)\) is disallowed. So path \(p_4 \rightarrow p_1 \rightarrow p_0\) is disallowed.

4. Global Path Constraints
   Due to local continuity constraints, certain portions of the \((p_x, p_y)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting
   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

   A path in \((p_x, p_y)\) plane can be represented in chain code. The value of the chain code is defined as following.
### shift (x, y) | chain code

| (1, 0) | 0 |
| (0, 1) | 1 |
| (1, 1) | 2 |
| (2, 1) | 3 |
| (1, 2) | 4 |
| (3, 1) | 5 |
| (3, 2) | 6 |
| (1, 3) | 7 |
| (2, 3) | 8 |

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In the following example, the observed data with 11 data points are mapped into the reference data with 9 data points.

```
py
  |
  * 8 7 *
  |
  * 4 * 6
  |
  1 2 3 5
  |
x--0--*--*-- px
```
The chain code that represents the path is

\((2 2 2 1 2 0 2 2 0 2 0)\)


**Parameters**  The function takes the following arguments:

- `dist` The distance of the optimal path.
- `dobs` The observed data array.
- `lobs` The length of the observed data array.
- `sobs` The scaling factor of the observed data array, where `actual_data = input_data * 2**(-scaling_factor)`.
- `state` Pointer to the internal state structure.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_SignalDTWScalar_Init_S16(3MLIB)`, `mlib_SignalDTWScalar_S16(3MLIB)`, `mlib_SignalDTWScalarPath_S16(3MLIB)`, `mlib_SignalDTWScalarFree_S16(3MLIB)`, attributes(5)
The `mlib_SignalDTWVector_F32()` function performs dynamic time warping on vector data.

Assume the reference data are
\[ r(y), \; y = 1, 2, \ldots, N \]
and the observed data are
\[ o(x), \; x = 1, 2, \ldots, M \]
the dynamic time warping is to find a mapping function (a path)
\[ p(i) = \{ px(i), py(i) \}, \; i = 1, 2, \ldots, Q \]
with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as
\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)), o(px(i))) \cdot m(px(i), py(i))
\]
where \( d(r, o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x, y) \) is the path weighting coefficient associated with path point \( (x, y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)
\[
L-1 \quad d(r, o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]

Using L2 norm (Euclidean distance)
\[
L-1 \quad d(r, o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } \]
where \( L \) is the length of each data vector.

To scalar data where \( L=1 \), the two norms are the same.
\[
d(r, o) = |r - o| = \sqrt{(r - o)^2}
\]
The constraints of dynamic time warping are:

1. **Endpoint constraints**
   
   \[ px(1) = 1 \]
   
   \[ 1 \leq py(1) \leq 1 + \delta \]

   and

   \[ px(N) = M \]
   
   \[ N - \delta \leq py(N) \leq N \]

2. **Monotonicity Conditions**

   \[ px(i) \leq px(i+1) \]
   
   \[ py(i) \leq py(i+1) \]

3. **Local Continuity Constraints**

   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   Itakura Type:

   ![Diagram of Itakura Type](attachment:image.png)

   Allowable paths are

   \[ p1 \rightarrow p0 \quad (1,0) \]
   
   \[ p2 \rightarrow p0 \quad (1,1) \]
   
   \[ p3 \rightarrow p0 \quad (1,2) \]

   Consecutive \((1,0)\) \((1,0)\) is disallowed. So path \(p4 \rightarrow p1 \rightarrow p0\) is disallowed.

4. **Global Path Constraints**

   Due to local continuity constraints, certain portions of the \((px, py)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. **Slope Weighting**

   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

A path in \((px, py)\) plane can be represented in chain code. The value of the chain code is defined as following.

---

**shift \((x, y)\) | chain code**

---
where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
py

| * 8 7 * |
| * 4 * 6 |
| 1 2 3 5 |
| x--0--*--*-- px
```
The chain code that represents the path is

\[(2 \ 2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


**Parameters**

The function takes the following arguments:

- `dist`  The distance of the optimal path.
- `dobs`  The observed data array.
- `lobs`  The length of the observed data array.
- `state`  Pointer to the internal state structure.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_SignalDTWVectorInit_F32(3MLIB)`, `mlib_SignalDTWVector_F32(3MLIB)`, `mlib_SignalDTWVectorPath_F32(3MLIB)`, `mlib_SignalDTWVectorFree_F32(3MLIB)`, attributes(5)
Name  mlib_SignalDTWVectorFree_S16, mlib_SignalDTWVectorFree_F32 – clean up for vector data

Synopsis  cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

void mlib_SignalDTWVectorFree_S16(void *state);
void mlib_SignalDTWVectorFree_F32(void *state);

Description  Each of these functions frees the internal state structure for dynamic time warping (DTW) of vector data. This function cleans up the internal state structure and releases all memory buffers.

Parameters  Each of the functions takes the following arguments:

state  Pointer to the internal state structure.

Return Values  None.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalDTWVectorInit_S16(3MLIB), mlib_SignalDTWVectorInit_F32(3MLIB), mlib_SignalDTWVector_S16(3MLIB), mlib_SignalDTWVector_F32(3MLIB), mlib_SignalDTWVectorPath_S16(3MLIB), mlib_SignalDTWVectorPath_F32(3MLIB), attributes(5)
Name mlib_SignalDTWVectorInit_F32 – initialization for vector data

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalDTWVectorInit_F32(void *state,
    const mlib_f32 **dref, mlib_s32 lref, mlib_s32 ndata,
    mlib_s32 dtype, mlib_s32 delta, mlib_s32 local,
    mlib_s32 slope);

Description The mlib_SignalDTWVectorInit_F32() function initializes the internal state structure for dynamic time warping (DTW) of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

Parameters The function takes the following arguments:

dref The reference data array.
lref The length of the reference data array.
ndata The length of each data vector.
dtype The type of distance metric between data vectors. MLIB_DTW_L1NORM for L1 norm of difference (sum of absolute difference). MLIB_DTW_L2NORM for L2 norm of difference (Euclidean distance).
delta The delta in the endpoint constraints.
local The type of the local continuity constraints. MLIB_DTW_ITAKURA for Itakura type constraints.
slope The type of the slope weighting. MLIB_DTW_NONE for no slope weighting.
state Pointer to the internal state structure.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalDTWVectorInit_F32(3MLIB), mlib_SignalDTWVector_F32(3MLIB), mlib_SignalDTWVectorPath_F32(3MLIB), mlib_SignalDTWVectorFree_F32(3MLIB), attributes(5)
The `mlib_SignalDTWVectorInit_S16()` function initializes the internal state structure for dynamic time warping (DTW) of vector data.

The init function performs internal state structure allocation and global initialization. Per DTW function call initialization is done in DTW function, so the same internal state structure can be reused for multiple DTW function calls.

### Parameters
The function takes the following arguments:

- **dref**  
  The reference data array.

- **lref**  
  The length of the reference data array.

- **ndata**  
  The length of each data vector.

- **dtype**  
  The type of distance metric between data vectors. `MLIB_DTW_L1NORM` for L1 norm of difference (sum of absolute difference). `MLIB_DTW_L2NORM` for L2 norm of difference (Euclidean distance).

- **sref**  
  The scaling factor of the reference data array, where `actual_data = input_data * 2**(-scaling_factor)`.

- **delta**  
  The delta in the endpoint constraints.

- **local**  
  The type of the local continuity constraints. `MLIB_DTW_ITAKURA` for Itakura type constraints.

- **slope**  
  The type of the slope weighting. `MLIB_DTW_NONE` for no slope weighting.

- **state**  
  Pointer to the internal state structure.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_SignalDTWVectorInit_S16(3MLIB), mlib_SignalDTWVector_S16(3MLIB),
mlib_SignalDTWVectorPath_S16(3MLIB), mlib_SignalDTWVectorFree_S16(3MLIB),
attributes(5)
The `mlib_SignalDTWVectorPath_F32()` function performs dynamic time warping on vector data.

Assume the reference data are
\[ r(y), \ y = 1, 2, \ldots, N \]
and the observed data are
\[ o(x), \ x = 1, 2, \ldots, M \]
the dynamic time warping is to find a mapping function (a path)
\[ p(i) = \{ px(i), py(i) \}, \ i = 1, 2, \ldots, 0 \]
with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as
\[
0 \sum_{i=1}^{Q} d(r(py(i)), o(px(i))) \ast m(px(i), py(i))
\]
where \( d(r, o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o; m(x, y) \)
is the path weighting coefficient associated with path point \( (x, y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)
\[
L-1 \sum_{i=0}^{L-1} |r(i) - o(i)|
\]
Using L2 norm (Euclidean distance)
\[
L-1 \sum_{i=0}^{L-1} \sqrt{\sum (r(i) - o(i))^2}
\]
where \( L \) is the length of each data vector.
To scalar data where \( L=1 \), the two norms are the same.
The constraints of dynamic time warping are:

1. Endpoint constraints

\[ px(1) = 1 \]
\[ 1 \leq py(1) \leq 1 + \delta \]

and

\[ px(Q) = M \]
\[ N - \delta \leq py(Q) \leq N \]

2. Monotonicity Conditions

\[ px(i) \leq px(i+1) \]
\[ py(i) \leq py(i+1) \]

3. Local Continuity Constraints

See Table 4.5 on page 211 in Rabiner and Juang’s book.

Itakura Type:

```
\[
\begin{array}{cccc}
  & * & * & * \\
| & p4 | p1 | p0 \\
| | | |
* & * & * & * \\
| | p2 |
| | | |
* & * & * & * & px \\
\end{array}
\]
```

Allowable paths are

- \( p1 \rightarrow p0 \) (1, 0)
- \( p2 \rightarrow p0 \) (1, 1)
- \( p3 \rightarrow p0 \) (1, 2)

Consecutive (1, 0) (1, 0) is disallowed. So path \( p4 \rightarrow p1 \rightarrow p0 \) is disallowed.

4. Global Path Constraints

Due to local continuity constraints, certain portions of the \((px, py)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

A path in \((px, py)\) plane can be represented in chain code. The value of the chain code is defined as following.
### Chain Code Table

<table>
<thead>
<tr>
<th>Shift (x, y)</th>
<th>Chain Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

The chain code describes the path of a segment from the start point to the end point. The numbers represent the direction of movement. For example, a move from (1, 0) to (0, 1) has a chain code of 0.

In the following example, the observed data with 11 points are mapped into the reference data with 9 points:

```
  py
  | * 8 7 * |
  | * 4 * 6 |
  | 1 2 3 5 |
  | x--0--*--*-- px |
```

Where `x` marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.
The chain code that represents the path is

\[(2 \ 2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


**Parameters**  The function takes the following arguments:

- \textit{dist}  The distance of the optimal path.
- \textit{path}  The optimal path.
- \textit{lpath}  The length of the optimal path.
- \textit{dobs}  The observed data array.
- \textit{lobs}  The length of the observed data array.
- \textit{state}  Pointer to the internal state structure.

**Return Values**  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_SignalDTWVectorInit_F32(3MLIB), mlib_SignalDTWVector_F32(3MLIB), mlib_SignalDTWVectorPath_F32(3MLIB), mlib_SignalDTWVectorFree_F32(3MLIB), attributes(5)
The `mlib_SignalDTWVectorPath_S16()` function performs dynamic time warping on vector data.

Assume the reference data are
\[ r(y), y=1,2,...,N \]
and the observed data are
\[ o(x), x=1,2,...,M \]
the dynamic time warping is to find a mapping function (a path)
\[ p(i) = \{px(i),py(i)\}, i=1,2,...,Q \]
with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as
\[ \text{dist} = \sum_{i=1}^{Q} d(r(py(i)),o(px(i))) \cdot m(px(i),py(i)) \]
where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o \); \( m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)
\[ L-1 \]
\[ d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)| \]

Using L2 norm (Euclidean distance)
\[ L-1 \]
\[ d(r,o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } \]

where \( L \) is the length of each data vector.
To scalar data where \( L=1 \), the two norms are the same.
The constraints of dynamic time warping are:

1. Endpoint constraints
   
   \[ \begin{align*}
   p_x(1) &= 1 \\
   1 &\leq p_y(1) \leq 1 + \Delta \\
   
   p_x(Q) &= M \\
   N - \Delta &\leq p_y(Q) \leq N
   \end{align*} \]

   and

2. Monotonicity Conditions
   
   \[ p_x(i) \leq p_x(i+1) \]
   \[ p_y(i) \leq p_y(i+1) \]

3. Local Continuity Constraints

   See Table 4.5 on page 211 in Rabiner and Juang’s book.

   Itakura Type:

   \[
   \begin{array}{c}
   | \ \\
   |p_4|p_1|p_0| \\
   | | | |
   |-------|
   |p_2| | |
   | | | |
   |-------|
   | | | | | | ||-- px
   |p_3|
   \end{array}
   \]

   Allowable paths are

   \[
   \begin{align*}
   p_1 \rightarrow p_0 & \quad (1, 0) \\
   p_2 \rightarrow p_0 & \quad (1, 1) \\
   p_3 \rightarrow p_0 & \quad (1, 2)
   \end{align*}
   \]

   Consecutive \((1, 0)(1, 0)\) is disallowed. So path \(p_4 \rightarrow p_1 \rightarrow p_0\) is disallowed.

4. Global Path Constraints

   Due to local continuity constraints, certain portions of the \((p_x, p_y)\) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting

   See Equation 4.150-3 on page 216 in Rabiner and Juang’s book.

   A path in \((p_x, p_y)\) plane can be represented in chain code. The value of the chain code is defined as following.
shift(x,y) | chain code
-----------------------------
(1,0)  | 0
(0,1)  | 1
(1,1)  | 2
(2,1)  | 3
(1,2)  | 4
(3,1)  | 5
(3,2)  | 6
(1,3)  | 7
(2,3)  | 8

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points

```
  9
  1
  2
  3
py
  | * 8 7 *
  | * 4 * 6
  | 1 2 3 5
  | x--0--*--*-- px
```
The chain code that represents the path is

\[(2 \ 2 \ 2 \ 1 \ 2 \ 0 \ 2 \ 2 \ 0 \ 2 \ 0)\]


**Parameters**  The function takes the following arguments:

- `dist`  The distance of the optimal path.
- `path`  The optimal path.
- `lpath`  The length of the optimal path.
- `dobs`  The observed data array.
- `lobs`  The length of the observed data array.
- `sobs`  The scaling factor of the observed data array, where `actual_data = input_data * 2**(-scaling_factor)`.
- `state`  Pointer to the internal state structure.

**Return Values**  The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_SignalDTWVectorInit_S16(3MLIB)`, `mlib_SignalDTWVector_S16(3MLIB)`, `mlib_SignalDTWVectorPath_S16(3MLIB)`, `mlib_SignalDTWVectorFree_S16(3MLIB)`, attributes(5)
The `mlib_SignalDTWVector_S16()` function performs dynamic time warping on vector data.

Assume the reference data are
\[ r(y), y=1,2,...,N \]
and the observed data are
\[ o(x), x=1,2,...,M \]
the dynamic time warping is to find a mapping function (a path)
\[ p(i) = \{px(i),py(i)\}, i=1,2,...,Q \]
with the minimum distance.

In K-best paths case, K paths with the K minimum distances are searched.

The distance of a path is defined as
\[
\text{dist} = \sum_{i=1}^{Q} d(r(py(i)), o(px(i))) \cdot m(px(i), py(i))
\]
where \( d(r,o) \) is the dissimilarity between data point/vector \( r \) and data point/vector \( o; m(x,y) \) is the path weighting coefficient associated with path point \( (x,y) \); \( N \) is the length of the reference data; \( M \) is the length of the observed data; \( Q \) is the length of the path.

Using L1 norm (sum of absolute differences)
\[
L-1 \quad d(r,o) = \sum_{i=0}^{L-1} |r(i) - o(i)|
\]
Using L2 norm (Euclidean distance)
\[
L-1 \quad d(r,o) = \sqrt{ \sum_{i=0}^{L-1} (r(i) - o(i))^2 } \]
where \( L \) is the length of each data vector.
To scalar data where \( L=1 \), the two norms are the same.
D(r,o) = |r - o| = \sqrt{(r - o)^2}

The constraints of dynamic time warping are:

1. Endpoint constraints
   \[ p_x(1) = 1 \]
   \[ 1 \leq p_y(1) \leq 1 + \delta \]

   and

   \[ p_x(Q) = M \]
   \[ N - \delta \leq p_y(Q) \leq N \]

2. Monotonicity Conditions
   \[ p_x(i) \leq p_x(i+1) \]
   \[ p_y(i) \leq p_y(i+1) \]

3. Local Continuity Constraints
   See Table 4.5 on page 211 in Rabiner and Juang's book.

   Itakura Type:

   \[ \begin{array}{c}
       p_y \\
       *----*----* \\
       |p4 |p1 |p0 \\
       | | | \\
       *----*----* \\
       | |p2 | \\
       | | | \\
       *----*----*--- px \\
       p3
   \end{array} \]

   Allowable paths are
   \[ p_1 \rightarrow p_0 \quad (1,0) \]
   \[ p_2 \rightarrow p_0 \quad (1,1) \]
   \[ p_3 \rightarrow p_0 \quad (1,2) \]

   Consecutive (1, 0) (1, 0) is disallowed. So path p4 \rightarrow p1 \rightarrow p0 is disallowed.

4. Global Path Constraints
   Due to local continuity constraints, certain portions of the (px, py) plane are excluded from the region the optimal warping path can traverse. This forms global path constraints.

5. Slope Weighting
   See Equation 4.150-3 on page 216 in Rabiner and Juang's book.

A path in (px, py) plane can be represented in chain code. The value of the chain code is defined as following.
### shift (x, y) | chain code

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>2</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>4</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>5</td>
</tr>
<tr>
<td>(3, 2)</td>
<td>6</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>7</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>8</td>
</tr>
</tbody>
</table>

---

**py**

```
* 8 7 *
* 4 6 *
1 2 3 5
```

```
x--0--*--*-- px
```

where x marks the start point of a path segment, the numbers are the values of the chain code for the segment that ends at the point.

In following example, the observed data with 11 data points are mapped into the reference data with 9 data points:

```
py
|
9 | * * * * * * * * *.*
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
|   | /  
|   | * * * * * * * * *.
1 | * * * * * * * * * |
```

---

mlib_SignalDTWVector_S16(3MLIB)
The chain code that represents the path is

\[(2 2 1 2 0 2 2 0 2 0)\]


**Parameters**

The function takes the following arguments:

- **dist**  
  The distance of the optimal path.
- **dobs**  
  The observed data array.
- **lobs**  
  The length of the observed data array.
- **sobs**  
  The scaling factor of the observed data array, where \(\text{actual\_data} = \text{input\_data} \times 2^{\text{-scaling\_factor}}\).
- **state**  
  Pointer to the internal state structure.

**Return Values**

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_SignalDTWVectorInit_S16(3MLIB), mlib_SignalDTWVector_S16(3MLIB), mlib_SignalDTWVectorPath_S16(3MLIB), mlib_SignalDTWVectorFree_S16(3MLIB), attributes(5)
mlib_SignalEmphasizeFree_S16_S16, mlib_SignalEmphasizeFree_S16S_S16S, mlib_SignalEmphasizeFree_F32_F32, mlib_SignalEmphasizeFree_F32S_F32S – clean up for signal pre-emphasizing

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalEmphasizeFree_S16_S16(void *filter);
void mlib_SignalEmphasizeFree_S16S_S16S(void *filter);
void mlib_SignalEmphasizeFree_F32_F32(void *filter);
void mlib_SignalEmphasizeFree_F32S_F32S(void *filter);

Description

Each of these functions releases the memory allocated for the internal state’s structure.

Parameters

Each of the functions takes the following arguments:

filter Internal filter structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalEmphasize_S16_S16_Sat(3MLIB),
mlib_SignalEmphasizeInit_S16_S16(3MLIB), attributes(5)
Name  mlib_SignalEmphasizeInit_S16_S16, mlib_SignalEmphasizeInit_S16S_S16S,
       mlib_SignalEmphasizeInit_F32_F32, mlib_SignalEmphasizeInit_F32S_F32S – initialization
       for signal pre-emphasizing

Synopsis cc [ flag... ] file... -o file [ library... ]
#include <mlib.h>

mlib_status mlib_SignalEmphasizeInit_S16_S16(void **filter,
       mlib_f32 alpha);
mlib_status mlib_SignalEmphasizeInit_S16S_S16S(void **filter,
       mlib_f32 alpha);
mlib_status mlib_SignalEmphasizeInit_F32_F32(void **filter,
       mlib_f32 alpha);
mlib_status mlib_SignalEmphasizeInit_F32S_F32S(void **filter,
       mlib_f32 alpha);

Description Each of these functions allocates memory for an internal filter structure and converts the filter
coefficients into the internal representation.

Parameters Each of the functions takes the following arguments:

  filter       Internal filter structure.
  alpha        Emphasizing coefficient. 0 < alpha < 1.0

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalEmphasize_S16_S16_Sat(3MLIB),
          mlib_SignalEmphasizeFree_S16_S16(3MLIB), attributes(5)
mlib_SignalEmphasize_S16_S16_Sat, mlib_SignalEmphasize_S16S_S16S_Sat, mlib_SignalEmphasize_F32_F32, mlib_SignalEmphasize_F32S_F32S – signal pre-emphasizing

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalEmphasize_S16_S16_Sat(mlib_s16 *dst,
const mlib_s16 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalEmphasize_S16S_S16S_Sat(mlib_s16 *dst,
const mlib_s16 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalEmphasize_F32_F32(mlib_f32 *dst,
const mlib_f32 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalEmphasize_F32S_F32S(mlib_f32 *dst,
const mlib_f32 *src, void *filter, mlib_s32 n);

Description

Each of these functions applies the preemphasizer to one signal packet and updates the filter states.

For monaural signals, the following equation is used:

dst[i] = src[i] - alpha*src[i - 1]

where i = 0, 1, ..., (n - 1); src[-1] = 0.

For stereo signals, the following equation is used:

dst[2*i + 1] = src[2*i + 1] - alpha*src[2*(i - 1) + 1]

where i = 0, 1, ..., (n - 1); src[-2] = src[-1] = 0.

Parameters

Each of the functions takes the following arguments:

dst Destination signal array.
src Source signal array.
filter Internal filter structure.
n Number of samples in the source signal array.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>ATTRIBUTE TYPE</td>
<td>ATTRIBUTE VALUE</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalEmphasizeFree_S16_S16(3MLIB),  
mlib_SignalEmphasizeInit_S16_S16(3MLIB), attributes(5)
mlib_SignalFFT_1, mlib_SignalFFT_1_S16_S16_Mod,
mlib_SignalFFT_1_S16C_S16C_Mod, mlib_SignalFFT_1_S16C_S16_Mod,
mlib_SignalFFT_1_S16_Mod, mlib_SignalFFT_1_S16C_Mod, mlib_SignalFFT_1_F32_F32,
mlib_SignalFFT_1_F32C_F32C, mlib_SignalFFT_1_F32C_F32, mlib_SignalFFT_1_F32,
mlib_SignalFFT_1_F32C, mlib_SignalFFT_1_D64_D64, mlib_SignalFFT_1_D64C_D64C,
mlib_SignalFFT_1_D64C_D64, mlib_SignalFFT_1_D64, mlib_SignalFFT_1_D64C – signal
Fast Fourier Transform (FFT)

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFFT_1_S16_S16_Mod(mlib_s16 *dstr, mlib_s16 *dsti,
const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);
mlib_status mlib_SignalFFT_1_S16C_S16C_Mod(mlib_s16 *dstrc,
const mlib_s16 *srcc, mlib_s32 order);
mlib_status mlib_SignalFFT_1_S16C_S16_Mod(mlib_s16 *dstrc,
const mlib_s16 *srcr, mlib_s32 order);
mlib_status mlib_SignalFFT_1_S16_Mod(mlib_s16 *srcdstr,
mlib_s16 *srcdsti, mlib_s32 order);
mlib_status mlib_SignalFFT_1_S16C_Mod(mlib_s16 *srcdstrc,
mlib_s32 order);
mlib_status mlib_SignalFFT_1_F32_F32(mlib_f32 *dstr,
mlib_f32 *dsti, const mlib_f32 *srcr, const mlib_f32 *srci,
mlib_s32 order);
mlib_status mlib_SignalFFT_1_F32C_F32C(mlib_f32 *dstrc,
const mlib_f32 *srcc, mlib_s32 order);
mlib_status mlib_SignalFFT_1_F32C_F32(mlib_f32 *dstrc,
const mlib_f32 *srcr, mlib_s32 order);
mlib_status mlib_SignalFFT_1_F32(mlib_f32 *srcdstr,
mlib_f32 *srcdsti, mlib_s32 order);
mlib_status mlib_SignalFFT_1_F32C(mlib_f32 *srcdstrc,
mlib_s32 order);
mlib_status mlib_SignalFFT_1_D64_D64(mlib_d64 *dstr,
mlib_d64 *dsti, const mlib_d64 *srcr, const mlib_d64 *srci,
mlib_s32 order);
mlib_status mlib_SignalFFT_1_D64C_D64C(mlib_d64 *dstrc,
const mlib_d64 *srcc, mlib_s32 order);
mlib_status mlib_SignalFFT_1_D64C_D64(mlib_d64 *dstrc,
const mlib_d64 *srcr, mlib_s32 order);
mlib_status mlib_SignalFFT_1_D64(mlib_d64 *srcdstr,
mlib_d64 *srcdsti, mlib_s32 order);
Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

\[
\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \cdot \exp(-j2\pi n k/N)
\]

and the following equation is used for inverse FFT (IFFT):

\[
\text{dst}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \cdot \exp(j2\pi n k/N)
\]

where

\[k = 0, 1, \ldots, (N - 1)\]
\[n = 0, 1, \ldots, (N - 1)\]
\[N = 2^{\text{order}}\]

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

\[
\text{mlib_Signal[FFT|IFFT]} \_\text{ScaleMode \_OutType \_InType \_OpMode()}
\]
\[
\text{mlib_Signal[FFT|IFFT]} \_\text{ScaleMode \_DataType \_OpMode()}
\]

The scaling factors \(C_1\) and \(C_2\) used in the equations are defined as follows:

- For ScaleMode = 1, \(C_1 = 1\) and \(C_2 = 2^{\text{order}}\).
- For ScaleMode = 2, \(C_1 = 2^{\text{order}}\) and \(C_2 = 1\).
- For ScaleMode = 3, \(C_1 = C_2 = 2^{\text{order}/2}\) when order is even, or \(C_1 = 2^{\text{order}+1}/2\) and \(C_2 = 2^{\text{order}/2}\) when order is odd.
- For ScaleMode = 4, \(C_1 = 2^{\text{P}}\) and \(C_2 = 2^{\text{Q}}\), where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**

Each function takes some of the following arguments:

- \(dstr\) Destination signal array that contains the real parts.
- \(dsti\) Destination signal array that contains the imaginary parts.
- \(srcr\) Source signal array that contains the real parts.
- \(srci\) Source signal array that contains the imaginary parts.


Source and destination signal array that contains the real parts.

Source and destination signal array that contains the imaginary parts.


Order of the transformation. The base-2 logarithm of the number of data samples.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalFFT_1(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_3(3MLIB), mlib_SignalFFT_4(3MLIB), mlib_SignalIFFT_1(3MLIB), mlib_SignalIFFT_2(3MLIB), mlib_SignalIFFT_3(3MLIB), mlib_SignalIFFT_4(3MLIB), attributes(5)
**Name**
mlib_SignalFFT_2, mlib_SignalFFT_2_S16_S16, mlib_SignalFFT_2_S16C_S16C, mlib_SignalFFT_2_S16C_S16, mlib_SignalFFT_2_S16_S16C, mlib_SignalFFT_2_F32_F32, mlib_SignalFFT_2_F32C_F32C, mlib_SignalFFT_2_F32C_F32, mlib_SignalFFT_2_F32, mlib_SignalFFT_2_F32C, mlib_SignalFFT_2_D64_D64, mlib_SignalFFT_2_D64C_D64C, mlib_SignalFFT_2_D64C_D64, mlib_SignalFFT_2_D64, mlib_SignalFFT_2_D64C – signal Fast Fourier Transform (FFT)

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFFT_2_S16_S16(mlib_s16 *dstr, mlib_s16 *dsti, const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16C_S16C(mlib_s16 *dstc, const mlib_s16 *srcc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16C_S16(mlib_s16 *dstc, const mlib_s16 *srcr, mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16(mlib_s16 *srcdstr, mlib_s16 *srcdsti, mlib_s32 order);

mlib_status mlib_SignalFFT_2_S16C(mlib_s16 *srcdstc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32_F32(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *srcr, const mlib_f32 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32C_F32C(mlib_f32 *dstc, const mlib_f32 *srcc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32C_F32(mlib_f32 *dstc, const mlib_f32 *srcr, mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32(mlib_f32 *srcdstr, mlib_f32 *srcdsti, mlib_s32 order);

mlib_status mlib_SignalFFT_2_F32C(mlib_f32 *srcdstc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64_D64(mlib_d64 *dstr, mlib_d64 *dsti, const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64C_D64C(mlib_d64 *dstc, const mlib_d64 *srcc, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64C_D64(mlib_d64 *dstc, const mlib_d64 *srcr, mlib_s32 order);

mlib_status mlib_SignalFFT_2_D64(mlib_d64 *srcdstr, mlib_d64 *srcdsti, mlib_s32 order);
mlib_status mlib_SignalFFT_2_D64C(mlib_d64 *srcdstc, mlib_s32 order);

**Description**  
Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

\[
\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \times \exp(-j2\pi n k / N)
\]

and the following equation is used for inverse FFT (IFFT):

\[
\text{dst}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \times \exp(j2\pi n k / N)
\]

where

- \( k = 0, 1, \ldots, (N - 1) \)
- \( n = 0, 1, \ldots, (N - 1) \)
- \( N = 2^{\text{order}} \)

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

- `mlib_SignalFFT[IFFT]_ScaleMode_OutType_InType_OpMode()`
- `mlib_SignalFFT[IFFT]_ScaleMode_DataType_OpMode()`

The scaling factors \( C_1 \) and \( C_2 \) used in the equations are defined as follows:

- For ScaleMode = 1, \( C_1 = 1 \) and \( C_2 = 2^{\text{order}} \).
- For ScaleMode = 2, \( C_1 = 2^{\text{order}} \) and \( C_2 = 1 \).
- For ScaleMode = 3, \( C_1 = C_2 = 2^{\text{order}}(\text{order}/2) \) when order is even, or \( C_1 = 2^{\text{order}(\text{order}+1)/2} \) and \( C_2 = 2^{\text{order}(\text{order}-1)/2} \) when order is odd.
- For ScaleMode = 4, \( C_1 = 2^{\text{P}} \) and \( C_2 = 2^{\text{Q}} \), where \( P \) and \( Q \) are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**  
Each function takes some of the following arguments:

- `dstr`  Destination signal array that contains the real parts.
- `dsti`  Destination signal array that contains the imaginary parts.
- `srcr`  Source signal array that contains the real parts.
- `srci`  Source signal array that contains the imaginary parts.
dstc Complex destination signal array. \( \text{dstc}[2*i] \) contains the real parts, and \( \text{dstc}[2*i+1] \) contains the imaginary parts.

srcc Complex source signal array. \( \text{srcc}[2*i] \) contains the real parts, and \( \text{srcc}[2*i+1] \) contains the imaginary parts.

srcdstr Source and destination signal array that contains the real parts.

srcdsti Source and destination signal array that contains the imaginary parts.

srcdstc Complex source and destination signal array. \( \text{srcdstc}[2*i] \) contains the real parts, and \( \text{srcdstc}[2*i+1] \) contains the imaginary parts.

order Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_SignalFFT_1(3MLIB)`, `mlib_SignalFFT_3(3MLIB)`, `mlib_SignalFFT_4(3MLIB)`, `mlib_SignalFFT_2(3MLIB)`, `mlib_SignalIFFT_1(3MLIB)`, `mlib_SignalIFFT_2(3MLIB)`, `mlib_SignalIFFT_3(3MLIB)`, `mlib_SignalIFFT_4(3MLIB)`, `attributes(5)`
Name
mlib_SignalFFT_3, mlib_SignalFFT_3_S16_S16_Mod,
mlib_SignalFFT_3_S16C_S16C_Mod, mlib_SignalFFT_3_S16C_S16_Mod,
mlib_SignalFFT_3_S16_Mod, mlib_SignalFFT_3_S16C_Mod, mlib_SignalFFT_3_F32_F32,
mlib_SignalFFT_3_F32C_F32C, mlib_SignalFFT_3_F32C_F32, mlib_SignalFFT_3_F32,
mlib_SignalFFT_3_F32C, mlib_SignalFFT_3_D64_D64, mlib_SignalFFT_3_D64C_D64C,
mlib_SignalFFT_3_D64C_D64, mlib_SignalFFT_3_D64, mlib_SignalFFT_3_D64C – signal
Fast Fourier Transform (FFT)

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFFT_3_S16_S16_Mod(mlib_s16 *dstr, mlib_s16 *dsti,
const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_3_S16C_S16C_Mod(mlib_s16 *dstc,
const mlib_s16 *srcc,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_S16C_S16_Mod(mlib_s16 *dstc,
const mlib_s16 *srcr,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_S16_S16_Mod(mlib_s16 *dstr,
mlib_s16 *srcre,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_S16C_S16C_Mod(mlib_s16 *srcdstr,
mlib_s16 *srcdsti,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_S16C_S16C_Mod(mlib_s16 *srcdstr,
mlib_s16 *srcdsti,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_F32_F32(mlib_f32 *dstr,
mlib_f32 *dsti,
const mlib_f32 *srcr, const mlib_f32 *srci, mlib_s32 order);

mlib_status mlib_SignalFFT_3_F32C_F32C(mlib_f32 *dstc,
const mlib_f32 *srcc,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_F32C_F32(mlib_f32 *dstr,
mlib_f32 *srcre,
const mlib_f32 *srcre, const mlib_f32 *srcre, mlib_s32 order);

mlib_status mlib_SignalFFT_3_F32C_F32C(mlib_f32 *srcdstr,
mlib_f32 *srcdsti,
mlib_s32 order);

mlib_status mlib_SignalFFT_3_F32C_F32(mlib_f32 *srcdstr,
mlib_f32 *srcre,
const mlib_f32 *srcre, const mlib_f32 *srcre, mlib_s32 order);

mlib_status mlib_SignalFFT_3_F32C_F32C(mlib_f32 *srcdstr,
mlib_f32 *srcre,
const mlib_f32 *srcre, const mlib_f32 *srcre, mlib_s32 order);

mlib_status mlib_SignalFFT_3_D64_D64(mlib_d64 *dstr,
mlib_d64 *dsti,
const mlib_d64 *srcr, const mlib_d64 *srcre, mlib_s32 order);
mlib_status mlib_SignalFFT_3_D64C_D64C(mlib_d64 *dstc,
    const mlib_d64 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalFFT_3_D64C_D64(mlib_d64 *dstc,
    const mlib_d64 *srcr,
    mlib_s32 order);

mlib_status mlib_SignalFFT_3_D64C(mlib_d64 *srcdstc,
    mlib_s32 order);

mlib_status mlib_SignalFFT_3_D64C(mlib_d64 *srcdstc,
    mlib_s32 order);

Description

Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

$$\frac{1}{N-1} \sum_{n=0}^{N-1} \text{src}[n] \cdot \exp(-j2\pi n k/N)$$

and the following equation is used for inverse FFT (IFFT):

$$\frac{1}{N-1} \sum_{k=0}^{N-1} \text{src}[k] \cdot \exp(j2\pi n k/N)$$

where

$$k = 0, 1, \ldots, (N - 1)$$
$$n = 0, 1, \ldots, (N - 1)$$
$$N = 2^\text{order}$$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

mlib_Signal[FFT|IFFT].ScaleMode_OutType_InType_OpMode()
mlib_Signal[FFT|IFFT].ScaleMode_DataType_OpMode()

The scaling factors $C_1$ and $C_2$ used in the equations are defined as follows:

- For ScaleMode = 1, $C_1 = 1$ and $C_2 = 2^\text{order}$.
- For ScaleMode = 2, $C_1 = 2^\text{order}$ and $C_2 = 1$.
- For ScaleMode = 3, $C_1 = C_2 = 2^\text{order}$ when order is even, or $C_1 = 2^{((\text{order}+1)/2)}$ and $C_2 = 2^{((\text{order}-1)/2)}$ when order is odd.
- For ScaleMode = 4, $C_1 = 2^{\text{P}}$ and $C_2 = 2^{\text{Q}}$, where P and Q are adaptive scaling factors and are generated by the functions.
For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**  
Each function takes some of the following arguments:

- `dstr`  
  Destination signal array that contains the real parts.

- `dsti`  
  Destination signal array that contains the imaginary parts.

- `srcr`  
  Source signal array that contains the real parts.

- `srci`  
  Source signal array that contains the imaginary parts.

- `dstc`  

- `srcc`  

- `srcdstr`  
  Source and destination signal array that contains the real parts.

- `srcdsti`  
  Source and destination signal array that contains the imaginary parts.

- `srcdstc`  

- `order`  
  Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_SignalFFT_3(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_4(3MLIB), mlib_SignalIFFT_1(3MLIB), mlib_SignalIFFT_2(3MLIB), mlib_SignalIFFT_3(3MLIB), mlib_SignalIFFT_4(3MLIB), attributes(5)`
Each of the functions in this group performs Fast Fourier Transform (FFT).

The following equation is used for forward FFT:

\[
\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \times e^{-j2\pi n k/N}
\]

and the following equation is used for inverse FFT (IFFT):

\[
\text{dst}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \times e^{j2\pi n k/N}
\]

where

- \(k = 0, 1, \ldots, (N - 1)\)
- \(n = 0, 1, \ldots, (N - 1)\)
- \(N = 2^{\text{order}}\)

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

\[
\text{mlib_Signal[FFT|IFFT]}_{\text{ScaleMode}}_{\text{OutType}}_{\text{InType}}_{\text{OpMode}}()
\]

The scaling factors \(C1\) and \(C2\) used in the equations are defined as follows:

- For ScaleMode = 1, \(C1 = 1\) and \(C2 = 2^{\text{order}}\). 

For ScaleMode = 2, \( C_1 = 2^{\text{order}} \) and \( C_2 = 1 \).

For ScaleMode = 3, \( C_1 = C_2 = 2^{*\left(\frac{\text{order}}{2}\right)} \) when order is even, or \( C_1 = 2^{*\left(\frac{\text{order}+1}{2}\right)} \) and \( C_2 = 2^{*\left(\frac{\text{order}-1}{2}\right)} \) when order is odd.

For ScaleMode = 4, \( C_1 = 2^{*\text{P}} \) and \( C_2 = 2^{*\text{Q}} \), where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**

Each function takes some of the following arguments:

- \( \text{dstr} \) Destination signal array that contains the real parts.
- \( \text{dsti} \) Destination signal array that contains the imaginary parts.
- \( \text{srcr} \) Source signal array that contains the real parts.
- \( \text{srci} \) Source signal array that contains the imaginary parts.
- \( \text{dstc} \) Complex destination signal array. \( \text{dstc}[2*i] \) contains the real parts, and \( \text{dstc}[2*i+1] \) contains the imaginary parts.
- \( \text{srcc} \) Complex source signal array. \( \text{srcc}[2*i] \) contains the real parts, and \( \text{srcc}[2*i+1] \) contains the imaginary parts.
- \( \text{srcdstr} \) Source and destination signal array that contains the real parts.
- \( \text{srcdsti} \) Source and destination signal array that contains the imaginary parts.
- \( \text{srcdstc} \) Complex source and destination signal array. \( \text{srcdstc}[2*i] \) contains the real parts, and \( \text{srcdstc}[2*i+1] \) contains the imaginary parts.
- \( \text{order} \) Order of the transformation. The base-2 logarithm of the number of data samples.
- \( \text{scale} \) Adaptive scaling factor.

**Return Values**

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_SignalFFT_1(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_3(3MLIB),
mlib_SignalIFFT_1(3MLIB), mlib_SignalIFFT_2(3MLIB), mlib_SignalIFFT_3(3MLIB),
mlib_SignalIFFT_4(3MLIB), attributes(5)
Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

\[
\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \times \text{window}[n] \times \exp(-j2\pi n k/N)
\]

and the IFFT functions use the following equation:
\[ \text{dst}[n] = \sum_{k=0}^{N-1} \text{src}[k] \cdot \text{window}[k] \cdot \exp(j2\pi n k/N) \]

where

\[ \begin{align*}
k &= 0, 1, \ldots, (N - 1) \\
n &= 0, 1, \ldots, (N - 1) \\
N &= 2^{\text{order}}
\end{align*} \]

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

\[
\text{mlib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()}
\]

\[
\text{mlib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()}
\]

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2**order.
- For ScaleMode = 2, C1 = 2**order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2**((order+1)/2) when order is even, or C1 = 2**((order/2) and C2 = 2**((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2**P and C2 = 2**Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- **dstr** Destination signal array that contains the real parts.
- **dsti** Destination signal array that contains the imaginary parts.
- **srcr** Source signal array that contains the real parts.
- **srci** Source signal array that contains the imaginary parts.
- **dstrc** Complex destination signal array. dstrc[2*i] contains the real parts, and dstrc[2*i+1] contains the imaginary parts.
- **srcdstr** Source and destination signal array that contains the real parts.
- **srcdsti** Source and destination signal array that contains the imaginary parts.
- **srcdstc** Complex source and destination signal array. srcdstc[2*i] contains the real parts, and srcdstc[2*i+1] contains the imaginary parts.
Window coefficient array with $2^{\text{order}}$ real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

Order of the transformation. The base-2 logarithm of the number of data samples.

Return Values Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalFFTW_2(3MLIB), mlib_SignalFFTW_3(3MLIB), mlib_SignalFFTW_4(3MLIB),
mlib_SignalIFFTW_1(3MLIB), mlib_SignalIFFTW_2(3MLIB),
mlib_SignalIFFTW_3(3MLIB), mlib_SignalIFFTW_4(3MLIB), attributes(5)
Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

$$
\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \times \text{window}[n] \times \exp(-j2\pi n k/N)
$$

and the IFFT functions use the following equation:
1 \[ \text{dst}[n] = \sum_{k=0}^{N-1} \text{src}[k] \times \text{window}[k] \times \exp(j2\pi n k / N) \]

where

\[ k = 0, 1, \ldots, (N - 1) \]
\[ n = 0, 1, \ldots, (N - 1) \]
\[ N = 2^{\text{order}} \]

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

\[ \text{mlib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()} \]
\[ \text{mlib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()} \]

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = \(2^{\text{order}}\).
- For ScaleMode = 2, C1 = \(2^{\text{order}}\) and C2 = 1.
- For ScaleMode = 3, C1 = C2 = \(2^{\text{order}/2}\) when order is even, or C1 = \(2^{{(\text{order}+1)/2}}\) and C2 = \(2^{{(\text{order}-1)/2}}\) when order is odd.
- For ScaleMode = 4, C1 = \(2^{\text{P}}\) and C2 = \(2^{\text{Q}}\), where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

Parameters  Each function takes some of the following arguments:

- \(\text{dstr}\)  Destination signal array that contains the real parts.
- \(\text{dsti}\)  Destination signal array that contains the imaginary parts.
- \(\text{srcr}\)  Source signal array that contains the real parts.
- \(\text{srci}\)  Source signal array that contains the imaginary parts.
- \(\text{dstc}\)  Complex destination signal array. \(\text{dstc}[2*1]\) contains the real parts, and \(\text{dstc}[2*1+1]\) contains the imaginary parts.
- \(\text{srcc}\)  Complex source signal array. \(\text{srcc}[2*1]\) contains the real parts, and \(\text{srcc}[2*1+1]\) contains the imaginary parts.
- \(\text{srcdstr}\)  Source and destination signal array that contains the real parts.
- \(\text{srcdsti}\)  Source and destination signal array that contains the imaginary parts.
- \(\text{srcdsrc}\)  Complex source and destination signal array. \(\text{srcdsrc}[2*1]\) contains the real parts, and \(\text{srcdsrc}[2*1+1]\) contains the imaginary parts.
mlib_SignalFFTW_2(3MLIB)

window Window coefficient array with $2^{\text{order}}$ real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

order Order of the transformation. The base-2 logarithm of the number of data samples.

Return Values Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalFFTW_1(3MLIB), mlib_SignalFFTW_3(3MLIB), mlib_SignalFFTW_4(3MLIB), mlib_SignalIFFTW_1(3MLIB), mlib_SignalIFFTW_2(3MLIB), mlib_SignalIFFTW_3(3MLIB), mlib_SignalIFFTW_4(3MLIB), attributes(5)
Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

\[
\text{dst[k]} = \frac{1}{N} \sum_{n=0}^{N-1} \text{src[n]} \cdot \text{window[n]} \cdot \exp(-j2\pi n k/N)
\]

and the IFFT functions use the following equation:
\[
\text{dst}[n] = \sum_{k=0}^{N-1} \text{src}[k] \times \text{window}[k] \times \exp(j2\pi n k / N)
\]

where

\[
k = 0, 1, \ldots, (N - 1)
\]

\[
n = 0, 1, \ldots, (N - 1)
\]

\[
N = 2^{*\text{order}}
\]

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

\[
\text{mlib_Signal[FFTW|IFFTW]}_{\text{ScaleMode}}_{\text{OutType}}_{\text{InType}}_{\text{OpMode}()}
\]

\[
\text{mlib_Signal[FFTW|IFFTW]}_{\text{ScaleMode}}_{\text{DataType}}_{\text{OpMode}()}
\]

The scaling factors \(C_1\) and \(C_2\) used in the equations are defined as follows:

- For ScaleMode = 1, \(C_1 = 1\) and \(C_2 = 2^{*\text{order}}\).
- For ScaleMode = 2, \(C_1 = 2^{*\text{order}}\) and \(C_2 = 1\).
- For ScaleMode = 3, \(C_1 = C_2 = 2^{*\text{(order/2)}}\) when order is even, or \(C_1 = 2^{*\text{(order+1)/2}}\) and \(C_2 = 2^{*\text{(order-1)/2}}\) when order is odd.
- For ScaleMode = 4, \(C_1 = 2^{*\text{P}}\) and \(C_2 = 2^{*\text{Q}}\), where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**

Each function takes some of the following arguments:

- \text{dstr} Destination signal array that contains the real parts.
- \text{dsti} Destination signal array that contains the imaginary parts.
- \text{srcr} Source signal array that contains the real parts.
- \text{srci} Source signal array that contains the imaginary parts.
- \text{dstc} Complex destination signal array. \text{dstc}[2*i] contains the real parts, and \text{dstc}[2*i+1] contains the imaginary parts.
- \text{src} Complex source signal array. \text{src}[2*i] contains the real parts, and \text{src}[2*i+1] contains the imaginary parts.
- \text{srcestr} Source and destination signal array that contains the real parts.
- \text{srcestsi} Source and destination signal array that contains the imaginary parts.
- \text{srcestc} Complex source and destination signal array. \text{srcestc}[2*i] contains the real parts, and \text{srcestc}[2*i+1] contains the imaginary parts.
window  Window coefficient array with \( 2^{\text{order}} \) real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

order   Order of the transformation. The base-2 logarithm of the number of data samples.

Return Values Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalFFTW_1(3MLIB), mlib_SignalFFTW_2(3MLIB), mlib_SignalFFTW_4(3MLIB),
           mlib_SignalIFFTW_1(3MLIB), mlib_SignalIFFTW_2(3MLIB),
           mlib_SignalIFFTW_3(3MLIB), mlib_SignalIFFTW_4(3MLIB), attributes(5)
mlib_SignalFFTW_4, mlib_SignalFFTW_4_S16_S16, mlib_SignalFFTW_4_S16C_S16C, mlib_SignalFFTW_4_S16C_S16, mlib_SignalFFTW_4_S16, mlib_SignalFFTW_4_S16C – signal Fast Fourier Transform with windowing (FFTW)

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalFFTW_4_S16_S16(mlib_s16 *dstr, mlib_s16 *dsti,
    const mlib_s16 *srcr, const mlib_s16 *srci, const mlib_s16 *window,
    mlib_s32 order, mlib_s32 *scale);
mlib_status mlib_SignalFFTW_4_S16C_S16C(mlib_s16 *dstc,
    const mlib_s16 *srcc, const mlib_s16 *window, mlib_s32 order,
    mlib_s32 *scale);
mlib_status mlib_SignalFFTW_4_S16C_S16(mlib_s16 *dstc,
    const mlib_s16 *srcc, const mlib_s16 *window, mlib_s32 order,
    mlib_s32 *scale);
mlib_status mlib_SignalFFTW_4_S16(mlib_s16 *srcdstr,
    mlib_s16 *srcdsti, const mlib_s16 *window, mlib_s32 order,
    mlib_s32 *scale);
mlib_status mlib_SignalFFTW_4_S16C(mlib_s16 *srcdstr,
    const mlib_s16 *window, mlib_s32 order, mlib_s32 *scale);
```

**Description**

Each of the functions in this group performs Fast Fourier Transform with windowing (FFTW).

The FFTW functions use the following equation:

\[
1 \quad \frac{N-1}{C1} \sum_{n=0}^{N-1} \{src[n] \times window[n] \times \exp(-j2*\pi*n*k/N)}
\]

and the IFFTW functions use the following equation:

\[
1 \quad \frac{N-1}{C2} \sum_{k=0}^{N-1} \{src[k] \times window[k] \times \exp(j2*\pi*n*k/N)}
\]

where

\[
k = 0, 1, \ldots, (N - 1) \\
n = 0, 1, \ldots, (N - 1) \\
N = 2^{\text{order}}
\]

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

```
mlib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mlib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```
The scaling factors $C_1$ and $C_2$ used in the equations are defined as follows:

- For $\text{ScaleMode} = 1$, $C_1 = 1$ and $C_2 = 2^{\text{order}}$.
- For $\text{ScaleMode} = 2$, $C_1 = 2^{\text{order}}$ and $C_2 = 1$.
- For $\text{ScaleMode} = 3$, $C_1 = C_2 = 2^{\frac{\text{order}}{2}}$ when $\text{order}$ is even, or $C_1 = 2^{\frac{\text{order}+1}{2}}$ and $C_2 = 2^{\frac{\text{order}-1}{2}}$ when $\text{order}$ is odd.
- For $\text{ScaleMode} = 4$, $C_1 = 2^{\text{P}}$ and $C_2 = 2^{\text{Q}}$, where $\text{P}$ and $\text{Q}$ are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**

Each function takes some of the following arguments:

- $dstr$ Destination signal array that contains the real parts.
- $dsti$ Destination signal array that contains the imaginary parts.
- $srcr$ Source signal array that contains the real parts.
- $srci$ Source signal array that contains the imaginary parts.
- $dstc$ Complex destination signal array. $\text{dstc}[2*i]$ contains the real parts, and $\text{dstc}[2*i+1]$ contains the imaginary parts.
- $srcc$ Complex source signal array. $\text{srcc}[2*i]$ contains the real parts, and $\text{srcc}[2*i+1]$ contains the imaginary parts.
- $srccdstr$ Source and destination signal array that contains the real parts.
- $srcdsti$ Source and destination signal array that contains the imaginary parts.
- $srccdsc$ Complex source and destination signal array. $\text{srccdsc}[2*i]$ contains the real parts, and $\text{srccdsc}[2*i+1]$ contains the imaginary parts.
- $window$ Window coefficient array with $2^{\text{order}}$ real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.
- $\text{order}$ Order of the transformation. The base-2 logarithm of the number of data samples.
- $\text{scale}$ Adaptive scaling factor.

**Return Values**

Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:
### ATTRIBUTE TYPE

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalFFTW_1(3MLIB), mlib_SignalFFTW_2(3MLIB), mlib_SignalFFTW_3(3MLIB), mlib_SignalIFFTW_1(3MLIB), mlib_SignalIFFTW_2(3MLIB), mlib_SignalIFFTW_3(3MLIB), mlib_SignalIFFTW_4(3MLIB), attributes(5)
mlib_SignalFIR_F32_F32

**Name**
mlib_SignalFIR_F32_F32 – Finite Impulse Response (FIR) filtering

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFIR_F32_F32(mlib_f32 *dst, const mlib_f32 *src,
    void *filter, mlib_s32 n);

**Description**
The `mlib_SignalFIR_F32_F32()` function applies the FIR filter to one signal packet and updates the filter state.

**Parameters**
The function takes the following arguments:

- `dst` Output signal array.
- `src` Input signal array.
- `filter` Internal filter structure.
- `n` Number of samples in the input signal array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)
**mlib_SignalFIR_F32S_F32S**

**Name**
mlib_SignalFIR_F32S_F32S – Finite Impulse Response (FIR) filtering

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFIR_F32S_F32S(mlib_f32 *dst, const mlib_f32 *src,
void *filter, mlib_s32 n);
```

**Description**
The mlib_SignalFIR_F32S_F32S() function applies the FIR filter to one signal packet and updates the filter state.

**Parameters**
The function takes the following arguments:
- **dst**
- **src**
- **filter**
  Internal filter structure.
- **n**
  Number of samples in the input signal array.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
attributes(5)
Name: mlib_SignalFIRFree_F32_F32 – Finite Impulse Response (FIR) filtering

Synopsis:
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalFIRFree_F32_F32(void *filter);
```

Description: The mlib_SignalFIRFree_F32_F32() function releases the memory allocated for the internal filter structure.

Parameters: The function takes the following arguments:
- `filter`: Internal filter structure.

Return Values: None.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: attributes(5)
mlib_SignalFIRFree_F32S_F32S – Finite Impulse Response (FIR) filtering

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalFIRFree_F32S_F32S(void *filter);
```

Description

The `mlib_SignalFIRFree_F32S_F32S()` function releases the memory allocated for the internal filter structure.

Parameters

The function takes the following arguments:

- `filter` Internal filter structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

attributes(5)
mlib_SignalFIRFree_S16_S16, mlib_SignalFIRFree_S16S_S16S – Finite Impulse Response (FIR) filtering

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalFIRFree_S16_S16(void *filter);
void mlib_SignalFIRFree_S16S_S16S(void *filter);

Description

Each of these functions releases the memory allocated for the internal filter structure.

Parameters

Each of the functions takes the following arguments:

filter Internal filter structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalFIR_S16_S16_Sat(3MLIB), mlib_SignalFIRInit_S16_S16(3MLIB), attributes(5)
mlib_SignalFIRInit_F32_F32() function allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

The function takes the following arguments:

- **filter**: Internal filter structure.
- **flt**: Filter coefficient array.
- **tap**: Taps of the filter.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
Name  

mlib_SignalFIRInit_F32S_F32S – Finite Impulse Response (FIR) filtering

Synopsis  

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFIRInit_F32S_F32S(void **filter,
const mlib_f32 *flt, mlib_s32 tap);

Description  

The mlib_SignalFIRInit_F32S_F32S() function allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

Parameters  

The function takes the following arguments:

- **filter**  
  Internal filter structure.

- **flt**  
  Filter coefficient array in stereo format. flt[2*i] contains Channel 0, and flt[2*i+1] contains Channel 1

- **tap**  
  Taps of the filter.

Return Values  

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  

attributes(5)
mlib_SignalFIRInit_S16_S16, mlib_SignalFIRInit_S16S_S16S – Finite Impulse Response (FIR) filtering

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFIRInit_S16_S16(void **filter,
   const mlib_f32 *flt, mlib_s32 tap);
mlib_status mlib_SignalFIRInit_S16S_S16S(void **filter,
   const mlib_f32 *flt, mlib_s32 tap);

Description
Each of these functions allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

Parameters
Each of the functions takes the following arguments:

- filter    Internal filter structure.
- flt       Filter coefficient array.
- tap       Taps of the filter.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalFIR_S16_S16_Sat(3MLIB), mlib_SignalFIRFree_S16_S16(3MLIB), attributes(5)
mlib_SignalFIR_S16_S16_Sat, mlib_SignalFIR_S16S_S16S_Sat – Finite Impulse Response (FIR) filtering

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalFIR_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, void *filter, mlib_s32 n);

mlib_status mlib_SignalFIR_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, void *filter, mlib_s32 n);

Description

Each of these functions applies the FIR filter to one signal packet and updates the filter state.

Parameters

Each of the functions takes the following arguments:

- **dst** Output signal array.
- **src** Input signal array.
- **filter** Internal filter structure.
- **n** Number of samples in the input signal array.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalFIRFree_S16_S16(3MLIB), mlib_SignalFIRInit_S16_S16(3MLIB), attributes(5)
The `mlib_SignalGaussNoise_F32()` function generates one packet of Gaussian noise and updates the internal state. The function takes the following arguments:

- `gnoise` - Generated Gaussian noise array.
- `state` - Internal state structure.
- `n` - Length of the generated Gaussian wave array in number of samples.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`attributes(5) `
mlib_SignalGaussNoiseFree_F32 – Gaussian noise generation

Synopsis

```c
#include <mlib.h>

void mlib_SignalGaussNoiseFree_F32(void *state);
```

Description

The `mlib_SignalGaussNoiseFree_F32()` function releases the memory allocated for the internal state’s structure.

Parameters

The function takes the following arguments:

- `state` - Internal state structure.

Return Values

None.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`attributes(5)`
The `mlib_SignalGaussNoiseFree_S16()` function releases the memory allocated for the internal state's structure.

### Parameters

- **state**: Internal state structure.

### Return Values

None.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalGaussNoise_S16(3MLIB)`, `mlib_SignalGaussNoiseInit_S16(3MLIB)`, `attributes(5)`
mlib_SignalGaussNoiseInit_F32 – Gaussian noise generation

Synopsis

```c
cc [ flag...] file... -lmlib [ library...] 
#include <mlib.h>

mlib_status mlib_SignalGaussNoiseInit_F32(void **state, mlib_f32 mag,
mlib_f32 mean, mlib_f32 stddev, mlib_f32 seed);
```

Description

The `mlib_SignalGaussNoiseInit_F32()` function allocates memory for an internal state
structure and converts the parameters into an internal representation.

Parameters

The function takes the following arguments:

- `state` Internal state structure.
- `mag` Magnitude of the Gaussian noise to be generated, in Q15 format.
- `mean` Mean of the Gaussian noise.
- `stddev` Standard deviation of the Gaussian noise.
- `seed` Seed value for the pseudorandom number generator.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`attributes(5)`
The `mlib_SignalGaussNoiseInit_S16()` function allocates memory for an internal state structure and converts the parameters into an internal representation.

### Parameters

- **state**: Internal state structure.
- **mag**: Magnitude of the Gaussian noise to be generated, in Q15 format.
- **mean**: Mean of the Gaussian noise.
- **stddev**: Standard deviation of the Gaussian noise.
- **seed**: Seed value for the pseudorandom number generator.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalGaussNoise_S16(3MLIB)`, `mlib_SignalGaussNoiseFree_S16(3MLIB)`, `attributes(5)`
The `mlib_SignalGaussNoise_S16()` function generates one packet of Gaussian noise and updates the internal state.

The function takes the following arguments:

- `gnoise`  Generated Gaussian noise array.
- `state`  Internal state structure.
- `n`  Length of the generated Gaussian wave array in number of samples.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See Also  `mlib_SignalGaussNoiseFree_S16(3MLIB), mlib_SignalGaussNoiseInit_S16(3MLIB), attributes(5)`. 

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
The `mlib_SignalGenBartlett_F32()` function generates the normalized coefficients of the Bartlett window.

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalGenBartlett_F32(mlib_f32 *window, mlib_s32 n);
```

**Description**
The `mlib_SignalGenBartlett_F32()` function generates the normalized coefficients of the Bartlett window.

**Parameters**
The function takes the following arguments:
- `window` - Generated window coefficient array.
- `n` - Length of window array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`attributes(5)"
The `mlib_SignalGenBartlett_S16()` function generates the normalized coefficients of the Bartlett window.

### Parameters

- **window**
  - Generated window coefficient array. The window coefficients are in Q15 format.
- **n**
  - Length of window array.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT- Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalGenHanning_S16(3MLIB)`, `mlib_SignalGenHamming_S16(3MLIB)`, `mlib_SignalGenBlackman_S16(3MLIB)`, `mlib_SignalGenKaiser_S16(3MLIB)`, [attributes(5)]
The `mlib_SignalGenBlackman_F32()` function generates the normalized coefficients of the Blackman window.

**Parameters**
The function takes the following arguments:

- `window` - Generated window coefficient array.
- `alpha` - Blackman window parameter. `-1 < alpha < 0`.
- `n` - Length of window array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`attributes(5)`
The `mlib_SignalGenBlackman_S16()` function generates the normalized coefficients of the Blackman window.

**Parameters**
- `window` Generated window coefficient array. The window coefficients are in Q15 format.
- `alpha` Blackman window parameter. `-1 < alpha < 0`.
- `n` Length of window array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_SignalGenBartlett_S16(3MLIB)`, `mlib_SignalGenHanning_S16(3MLIB)`,
- `mlib_SignalGenHamming_S16(3MLIB)`, `mlib_SignalGenKaiser_S16(3MLIB)`,
The `mlib_SignalGenHamming_F32()` function generates the normalized coefficients of the Hamming window. The function takes the following arguments:

- `window` - Generated window coefficient array.
- `n` - Length of window array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

attributes(5)
The `mlib_SignalGenHamming_S16()` function generates the normalized coefficients of the Hamming window.

**Parameters**
The function takes the following arguments:

- `window`  
  Generated window coefficient array. The window coefficients are in Q15 format.
- `n`  
  Length of window array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_SignalGenBartlett_S16(3MLIB), mlib_SignalGenHanning_S16(3MLIB),
mlib_SignalGenBlackman_S16(3MLIB), mlib_SignalGenKaiser_S16(3MLIB),
attributes(5)`
The `mlib_SignalGenHanning_F32()` function generates the normalized coefficients of the Hanning window.

The function takes the following arguments:

- `window`: Generated window coefficient array. The window coefficients are in Q15 format.
- `n`: Length of window array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)
# mlib_SignalGenHanning_S16

## Name
mlib_SignalGenHanning_S16 – Hanning window generation

## Synopsis
```c
#include <mlib.h>

mlib_status mlib_SignalGenHanning_S16(mlib_s16 *window, mlib_s32 n);
```

## Description
The `mlib_SignalGenHanning_S16()` function generates the normalized coefficients of the Hanning window.

## Parameters
- **window**: Generated window coefficient array. The window coefficients are in Q15 format.
- **n**: Length of window array.

## Return Values
The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_SignalGenBartlett_S16(3MLIB)`, `mlib_SignalGenHamming_S16(3MLIB)`, `mlib_SignalGenBlackman_S16(3MLIB)`, `mlib_SignalGenKaiser_S16(3MLIB)`, `attributes(5)`
The `mlib_SignalGenKaiser_F32()` function generates the normalized coefficients of the Kaiser window.

### Parameters
The function takes the following arguments:

- `window` - Generated window coefficient array. The window coefficients are in Q15 format.
- `beta` - Kaiser window parameter.
- `n` - Length of window array.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
attributes(5)
### Name
mlib_SignalGenKaiser_S16 – Kaiser window generation

### Synopsis
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalGenKaiser_S16(mlib_s16 *window, mlib_f32 beta,
                                    mlib_s32 n);
```

### Description
The `mlib_SignalGenKaiser_S16()` function generates the normalized coefficients of the Kaiser window.

### Parameters
The function takes the following arguments:
- `window`: Generated window coefficient array. The window coefficients are in Q15 format.
- `beta`: Kaiser window parameter.
- `n`: Length of window array.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_SignalGenBartlett_S16(3MLIB)`, `mlib_SignalGenHanning_S16(3MLIB)`, `mlib_SignalGenHamming_S16(3MLIB)`, `mlib_SignalGenBlackman_S16(3MLIB)`, `attributes(5)`
**Name**  
mlib_SignalIFFT_1, mlib_SignalIFFT_1_S16_S16, mlib_SignalIFFT_1_S16C_S16C,  
mlib_SignalIFFT_1_S16_S16C, mlib_SignalIFFT_1_S16, mlib_SignalIFFT_1_S16C,  
mlib_SignalIFFT_1_F32_F32, mlib_SignalIFFT_1_F32C_F32C,  
mlib_SignalIFFT_1_F32_F32C, mlib_SignalIFFT_1_F32, mlib_SignalIFFT_1_F32C,  
mlib_SignalIFFT_1_D64_D64, mlib_SignalIFFT_1_D64C_D64C,  
mlib_SignalIFFT_1_D64_D64C, mlib_SignalIFFT_1_D64, mlib_SignalIFFT_1_D64C –  
signalInvereseFastFourierTransform (IFFT)

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

```c
mlib_status mlib_SignalIFFT_1_S16_S16(mlib_s16 *dstr, mlib_s16 *dsti,  
   const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);
mlib_status mlib_SignalIFFT_1_S16C_S16C(mlib_s16 *dstc,  
   const mlib_s16 *srcc,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_S16_S16C(mlib_s16 *dstr,  
   const mlib_s16 *srcc,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_S16(mlib_s16 *srcdstr,  
   mlib_s16 *srcdsti,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_S16C(mlib_s16 *srcdstc,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_F32_F32(mlib_f32 *dstr, mlib_f32 *dsti,  
   const mlib_f32 *srcr,  
   const mlib_f32 *srci, mlib_s32 order);
mlib_status mlib_SignalIFFT_1_F32C_F32C(mlib_f32 *dstc,  
   const mlib_f32 *srcc,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_F32_F32C(mlib_f32 *dstr,  
   const mlib_f32 *srcc,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_F32C(mlib_f32 *srcdstc,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_F32(mlib_f32 *srcdstr,  
   mlib_f32 *srcdsti,  
   mlib_s32 order);
mlib_status mlib_SignalIFFT_1_D64_D64(mlib_d64 *dstr, mlib_d64 *dsti,  
   const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);
```

Multimedia Library Functions - Part 4
mlib_status mlib_SignalIFFT_1_D64C_D64C(mlib_d64 *dstc,  
    const mlib_d64 *srcc,  
    mlib_s32 order);

mlib_status mlib_SignalIFFT_1_D64_D64C(mlib_d64 *dstr,  
    const mlib_d64 *srcc,  
    mlib_s32 order);

mlib_status mlib_SignalIFFT_1_D64(mlib_d64 *srcdstr,  
    mlib_d64 *srcdsti,  
    mlib_s32 order);

mlib_status mlib_SignalIFFT_1_D64C(mlib_d64 *srcdstc,  
    mlib_s32 order);

Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

\[
\frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \exp(-j2\pi n k/N)\]

and the following equation is used for inverse FFT (IFFT):

\[
\frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \exp(j2\pi n k/N)\]

where

\[k = 0, 1, \ldots, (N - 1)\]
\[n = 0, 1, \ldots, (N - 1)\]
\[N = 2^{\text{order}}\]

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

mlib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()

mlib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2**order.
- For ScaleMode = 2, C1 = 2**order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2**((order/2) when order is even, or C1 = 2**((order+1)/2) and C2 = 2**((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2**P and C2 = 2**Q, where P and Q are adaptive scaling factors and are generated by the functions.
For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**  
Each function takes some of the following arguments:
- $dstr$  
  Destination signal array that contains the real parts.
- $dsti$  
  Destination signal array that contains the imaginary parts.
- $srcr$  
  Source signal array that contains the real parts.
- $srci$  
  Source signal array that contains the imaginary parts.
- $dstc$  
  Complex destination signal array. $\text{dstc}[2*i]$ contains the real parts, and $\text{dstc}[2*i+1]$ contains the imaginary parts.
- $srcc$  
  Complex source signal array. $\text{srcc}[2*i]$ contains the real parts, and $\text{srcc}[2*i+1]$ contains the imaginary parts.
- $srcdstr$  
  Source and destination signal array that contains the real parts.
- $srcdsti$  
  Source and destination signal array that contains the imaginary parts.
- $srcdstc$  
  Complex source and destination signal array. $\text{srcdstc}[2*i]$ contains the real parts, and $\text{srcdstc}[2*i+1]$ contains the imaginary parts.
- $order$  
  Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values**  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalFFT_1(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_3(3MLIB), mlib_SignalFFT_4(3MLIB), mlib_SignalIFFT_2(3MLIB), mlib_SignalIFFT_3(3MLIB), mlib_SignalIFFT_4(3MLIB), attributes(5)
mlib_SignalIFFT_2(3MLIB)

Name

mlib_SignalIFFT_2, mlib_SignalIFFT_2_S16_S16_Mod,
mlib_SignalIFFT_2_S16C_S16C_Mod, mlib_SignalIFFT_2_S16_S16C_Mod,
mlib_SignalIFFT_2_S16_Mod, mlib_SignalIFFT_2_S16C_Mod,
mlib_SignalIFFT_2_F32_F32, mlib_SignalIFFT_2_F32C_F32C,
mlib_SignalIFFT_2_F32_F32C, mlib_SignalIFFT_2_F32, mlib_SignalIFFT_2_F32C,
mlib_SignalIFFT_2_D64_D64, mlib_SignalIFFT_2_D64C_D64C,
mlib_SignalIFFT_2_D64_D64C, mlib_SignalIFFT_2_D64, mlib_SignalIFFT_2_D64C

Signal Inverse Fast Fourier Transform (IFFT)

Synopsis

c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalIFFT_2_S16_S16_Mod(mlib_s16 *dstr,
    mlib_s16 *dsti,
    const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);

mlib_status mlib_SignalIFFT_2_S16C_S16C_Mod(mlib_s16 *dstc,
    const mlib_s16 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_S16_S16C_Mod(mlib_s16 *dstr,
    const mlib_s16 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_S16_Mod(mlib_s16 *srcdstr,
    mlib_s16 *srcdsti,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_S16C_Mod(mlib_s16 *srccdstc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_F32_F32(mlib_f32 *dstr,
    mlib_f32 *dsti,
    const mlib_f32 *srcr, const mlib_f32 *srci, mlib_s32 order);

mlib_status mlib_SignalIFFT_2_F32C_F32C(mlib_f32 *dstc,
    const mlib_f32 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_F32_F32C(mlib_f32 *dstr,
    const mlib_f32 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_F32C(mlib_f32 *srcdstr,
    mlib_f32 *srcdsti,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_F32C(mlib_f32 *srccdstc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_D64_D64(mlib_d64 *dstr,
    mlib_d64 *dsti,
    const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);
mlib_status mlib_SignalIFFT_2_D64C_D64C(mlib_d64 *dstc,
    const mlib_d64 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_D64_D64C(mlib_d64 *dstr,
    const mlib_d64 *srcc,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_D64(mlib_d64 *srcdstr,
    mlib_d64 *srcdsti,
    mlib_s32 order);

mlib_status mlib_SignalIFFT_2_D64C(mlib_d64 *srcdstc,
    mlib_s32 order);

**Description**  Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

\[
\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \times \exp(-j2\pi n k/N)
\]

and the following equation is used for inverse FFT (IFFT):

\[
\text{dst}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \times \exp(j2\pi n k/N)
\]

where

- \( k = 0, 1, ..., (N - 1) \)
- \( n = 0, 1, ..., (N - 1) \)
- \( N = 2^{\text{order}} \)

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

mlib_Signal[FFT|IFFT].ScaleMode_OutType_InType_OpMode()
mlib_Signal[FFT|IFFT].ScaleMode_DataType_OpMode()

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, \( C1 = 1 \) and \( C2 = 2^{\text{order}} \).
- For ScaleMode = 2, \( C1 = 2^{\text{order}} \) and \( C2 = 1 \).
- For ScaleMode = 3, \( C1 = C2 = 2^{\text{order}} \) when order is even, or \( C1 = 2^{((\text{order}+1)/2)} \) and \( C2 = 2^{((\text{order}-1)/2)} \) when order is odd.
- For ScaleMode = 4, \( C1 = 2^{\times P} \) and \( C2 = 2^{\times Q} \), where \( P \) and \( Q \) are adaptive scaling factors and are generated by the functions.
For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- **dstr** Destination signal array that contains the real parts.
- **dsti** Destination signal array that contains the imaginary parts.
- **srcr** Source signal array that contains the real parts.
- **srci** Source signal array that contains the imaginary parts.
- **dstc** Complex destination signal array. \( \text{dstc}[2*i] \) contains the real parts, and \( \text{dstc}[2*i+1] \) contains the imaginary parts.
- **srcc** Complex source signal array. \( \text{srcc}[2*i] \) contains the real parts, and \( \text{srcc}[2*i+1] \) contains the imaginary parts.
- **srcdstr** Source and destination signal array that contains the real parts.
- **srcdsti** Source and destination signal array that contains the imaginary parts.
- **srcdstc** Complex source and destination signal array. \( \text{srcdstc}[2*i] \) contains the real parts, and \( \text{srcdstc}[2*i+1] \) contains the imaginary parts.
- **order** Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTEVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_SignalFFT_1(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_3(3MLIB), mlib_SignalFFT_4(3MLIB), mlib_SignalIFFT_1(3MLIB), mlib_SignalIFFT_3(3MLIB), mlib_SignalIFFT_4(3MLIB), attributes(5)
**Name**
mlib_SignalIFFT_3, mlib_SignalIFFT_3_S16_S16_Mod,
mlib_SignalIFFT_3_S16C_S16C_Mod, mlib_SignalIFFT_3_S16_S16C_Mod,
mlib_SignalIFFT_3_S16_Mod, mlib_SignalIFFT_3_S16C_Mod,
mlib_SignalIFFT_3_F32_F32, mlib_SignalIFFT_3_F32C_F32C,
mlib_SignalIFFT_3_F32_F32C, mlib_SignalIFFT_3_F32C, mlib_SignalIFFT_3_F32C,
mlib_SignalIFFT_3_D64_D64, mlib_SignalIFFT_3_D64C_D64C,
mlib_SignalIFFT_3_D64_D64C, mlib_SignalIFFT_3_D64, mlib_SignalIFFT_3_D64C –
signalInverseFastFourierTransform (IFFT)

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include "mlib.h"

```c
mlib_status mlib_SignalIFFT_3_S16_S16_Mod(mlib_s16 *dstr, mlib_s16 *dsti,
const mlib_s16 *srcr, const mlib_s16 *srci, mlib_s32 order);

mlib_status mlib_SignalIFFT_3_S16C_S16C_Mod(mlib_s16 *dstr,
const mlib_s16 *srcc,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_S16_S16C_Mod(mlib_s16 *dstr,
const mlib_s16 *srcc,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_S16_Mod(mlib_s16 *srcdstr,
mlib_s16 *srccdsti,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_S16C_Mod(mlib_s16 *srcdstc,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_F32_F32(mlib_f32 *dstr,
mlib_f32 *dsti,
const mlib_f32 *srcr, const mlib_f32 *srci,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_F32C_F32C(mlib_f32 *dstr,
const mlib_f32 *srcc,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_F32_F32C(mlib_f32 *dstr,
const mlib_f32 *srcc,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_F32C(mlib_f32 *srcdstr,
mlib_f32 *srccdsti,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_F32C(mlib_f32 *srcdstc,
mlib_s32 order);

mlib_status mlib_SignalIFFT_3_D64_D64(mlib_d64 *dstr,
mlib_d64 *dsti,
const mlib_d64 *srcr, const mlib_d64 *srci, mlib_s32 order);
```
mlib_status mlib_SignalIFFT_3_D64C_D64C(mlib_d64 *dstc,  
    const mlib_d64 *srcc,  
    mlib_s32 order);

mlib_status mlib_SignalIFFT_3_D64_D64C(mlib_d64 *dstr,  
    const mlib_d64 *srcc,  
    mlib_s32 order);

mlib_status mlib_SignalIFFT_3_D64(mlib_d64 *srcdst,  
    mlib_d64 *srcdsti,  
    mlib_s32 order);

mlib_status mlib_SignalIFFT_3_D64C(mlib_d64 *srcdstc, mlib_s32 order);

Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

\[ dst[k] = \frac{1}{N} \sum_{n=0}^{N-1} src[n] \cdot \exp(-j2\pi n k/N) \]  

C1

and the following equation is used for inverse FFT (IFFT):

\[ dst[n] = \frac{1}{N} \sum_{k=0}^{N-1} src[k] \cdot \exp(j2\pi n k/N) \]  

C2

where

k = 0, 1, ..., (N - 1)
n = 0, 1, ..., (N - 1)
N = 2**order

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

mlib_Signal[FFT|IFFT]_ScaleMode_OutType_InType_OpMode()  
mlib_Signal[FFT|IFFT]_ScaleMode_DataType_OpMode()

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2**order.
- For ScaleMode = 2, C1 = 2**order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2**((order/2) when order is even, or C1 = 2**((order+1)/2) and C2 = 2**((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2**P and C2 = 2**Q, where P and Q are adaptive scaling factors and are generated by the functions.
For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- **dstr** Destination signal array that contains the real parts.
- **dsti** Destination signal array that contains the imaginary parts.
- **srcr** Source signal array that contains the real parts.
- **srci** Source signal array that contains the imaginary parts.
- **dstc** Complex destination signal array. 
  \[ \text{dstc}[2*i] \] contains the real parts, and 
  \[ \text{dstc}[2*i+1] \] contains the imaginary parts.
- **srcc** Complex source signal array. 
  \[ \text{srcc}[2*i] \] contains the real parts, and 
  \[ \text{srcc}[2*i+1] \] contains the imaginary parts.
- **srcdstc** Source and destination signal array that contains the real parts.
- **order** Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** 
mlib_SignalFFT_1(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_3(3MLIB), mlib_SignalFFT_4(3MLIB), mlib_SignalIFFT_1(3MLIB), mlib_SignalIFFT_2(3MLIB), mlib_SignalIFFT_3(3MLIB), mlib_SignalIFFT_4(3MLIB), attributes(5)
Each of the functions in this group performs Inverse Fast Fourier Transform (IFFT).

The following equation is used for forward FFT:

$$\text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \cdot \exp(-j2\pi n k/N)$$

and the following equation is used for inverse FFT (IFFT):

$$\text{dst}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \cdot \exp(j2\pi n k/N)$$

where

- $k = 0, 1, \ldots, (N - 1)$
- $n = 0, 1, \ldots, (N - 1)$
- $N = 2^{\text{order}}$

The signal FFT/IFFT functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

- `mlib_SignalFFT[IFT]_ScaleMode_OutType_InType_OpMode()`
- `mlib_SignalFFT[IFT]_ScaleMode_DataType_OpMode()`

The scaling factors $C_1$ and $C_2$ used in the equations are defined as follows:
For ScaleMode = 1, \( C_1 = 1 \) and \( C_2 = 2^{\text{order}} \).

For ScaleMode = 2, \( C_1 = 2^{\text{order}} \) and \( C_2 = 1 \).

For ScaleMode = 3, \( C_1 = C_2 = 2^{\left(\text{order}/2\right)} \) when \text{order} is even, or \( C_1 = 2^{\left((\text{order}+1)/2\right)} \) and \( C_2 = 2^{\left((\text{order}-1)/2\right)} \) when \text{order} is odd.

For ScaleMode = 4, \( C_1 = 2^{\text{P}} \) and \( C_2 = 2^{\text{Q}} \), where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- **dstr** Destination signal array that contains the real parts.
- **dsti** Destination signal array that contains the imaginary parts.
- **srcr** Source signal array that contains the real parts.
- **srci** Source signal array that contains the imaginary parts.
- **dsrc** Complex destination signal array. \( \text{dsrc}[2*i] \) contains the real parts, and \( \text{dsrc}[2*i+1] \) contains the imaginary parts.
- **srcc** Complex source signal array. \( \text{srcc}[2*i] \) contains the real parts, and \( \text{srcc}[2*i+1] \) contains the imaginary parts.
- **srcdstr** Source and destination signal array that contains the real parts.
- **srcdstr** Source and destination signal array that contains the imaginary parts.
- **srcdstc** Complex source and destination signal array. \( \text{srcdstr}[2*i] \) contains the real parts, and \( \text{srcdstr}[2*i+1] \) contains the imaginary parts.
- **order** Order of the transformation. The base-2 logarithm of the number of data samples.
- **scale** Adaptive scaling factor.

**Return Values** The function returns MLLIB_SUCCESS if successful. Otherwise it returns MLLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_SignalFFT_1(3MLIB), mlib_SignalFFT_2(3MLIB), mlib_SignalFFT_3(3MLIB),
mlib_SignalFFT_4(3MLIB), mlib_SignalIFFT_1(3MLIB), mlib_SignalIFFT_2(3MLIB),
mlib_SignalIFFT_3(3MLIB), attributes(5)
Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:
The signal FFTW/IFFTW functions can be categorized into four groups according to the `ScaleMode` in the function names in the following form:

```
mlib_Signal[FFTW|IFFTW]_ScaleMode_OutType_InType_OpMode()
mlib_Signal[FFTW|IFFTW]_ScaleMode_DataType_OpMode()
```

The scaling factors $C_1$ and $C_2$ used in the equations are defined as follows:

- For `ScaleMode = 1`, $C_1 = 1$ and $C_2 = 2^{**\text{order}}$.
- For `ScaleMode = 2`, $C_1 = 2^{**\text{order}}$ and $C_2 = 1$.
- For `ScaleMode = 3`, $C_1 = C_2 = 2^{**(\text{order/2})}$ when `order` is even, or
  $C_1 = 2^{**(\text{order+1)/2}}$ and $C_2 = 2^{**(\text{order-1)/2}}$ when `order` is odd.
- For `ScaleMode = 4`, $C_1 = 2^{**P}$ and $C_2 = 2^{**Q}$, where $P$ and $Q$ are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters** Each function takes some of the following arguments:

- $\text{dstr}$ Destination signal array that contains the real parts.
- $\text{dsti}$ Destination signal array that contains the imaginary parts.
- $\text{srcr}$ Source signal array that contains the real parts.
- $\text{srci}$ Source signal array that contains the imaginary parts.
- $\text{dsrc}$ Complex destination signal array. $\text{dsrc}[2*1]$ contains the real parts, and $\text{dsrc}[2*1+1]$ contains the imaginary parts.
- $\text{srcc}$ Complex source signal array. $\text{srcc}[2*1]$ contains the real parts, and $\text{srcc}[2*1+1]$ contains the imaginary parts.
**mlib_SignalIFFTW_1(3MLIB)**

**srcdstr** Source and destination signal array that contains the real parts.

**srcdsti** Source and destination signal array that contains the imaginary parts.

**srcdstc** Complex source and destination signal array. `srcdstc[2*i]` contains the real parts, and `srcdstc[2*i+1]` contains the imaginary parts.

**window** Window coefficient array with `2**order` real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

**order** Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE Tipo</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_SignalFFTW_1(3MLIB), mlib_SignalFFTW_2(3MLIB), mlib_SignalFFTW_3(3MLIB), mlib_SignalFFTW_4(3MLIB), mlib_SignalIFFTW_2(3MLIB), mlib_SignalIFFTW_3(3MLIB), mlib_SignalIFFTW_4(3MLIB), attributes(5)`
Multimedia Library Functions - Part 5
Name  
mllib_SignalIFFTW_2, mllib_SignalIFFTW_2_S16_S16_Mod,  
mllib_SignalIFFTW_2_S16C_S16C_Mod, mllib_SignalIFFTW_2_S16_S16C_Mod,  
mllib_SignalIFFTW_2_S16_Mod, mllib_SignalIFFTW_2_S16C_Mod,  
mllib_SignalIFFTW_2_F32_F32, mllib_SignalIFFTW_2_F32C_F32C,  
mllib_SignalIFFTW_2_F32_F32C, mllib_SignalIFFTW_2_F32, mllib_SignalIFFTW_2_F32C –  
signal Inverse Fast Fourier Transform with windowing (IFFTW)

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

  mlib_status mlib_SignalIFFTW_2_S16_S16_Mod(mlib_s16 *dstr,  
   mlib_s16 *dsti,  
   const mlib_s16 *srcr, const mlib_s16 *srci, const mlib_s16 *window,  
   mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_S16C_S16C_Mod(mlib_s16 *dstr,  
   const mlib_s16 *srcc,  
   const mlib_s16 *window, mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_S16_S16C_Mod(mlib_s16 *dstr,  
   const mlib_s16 *srcc,  
   const mlib_s16 *window, mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_S16_Mod(mlib_s16 *srcdstr,  
   mlib_s16 *srcdsti,  
   const mlib_s16 *window, mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_S16C_Mod(mlib_s16 *srcdstc,  
   const mlib_s16 *window, mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_F32_F32(mlib_f32 *dstr,  
   mlib_f32 *dsti,  
   const mlib_f32 *srcr, const mlib_f32 *srci, const mlib_f32 *window,  
   mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_F32C_F32C(mlib_f32 *dstr,  
   const mlib_f32 *srcc,  
   const mlib_f32 *window, mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_F32_F32C(mlib_f32 *dstr,  
   const mlib_f32 *srcc,  
   const mlib_f32 *window, mlib_s32 order);

  mlib_status mlib_SignalIFFTW_2_F32C_F32C(mlib_f32 *srcc,  
   const mlib_f32 *window, mlib_s32 order);
### Description
Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

\[
\text{dst}[k] = \frac{1}{N-1} \sum_{n=0}^{N-1} \text{src}[n] \times \text{window}[n] \times \exp(-j2\pi n k/N)
\]

and the IFFT W functions use the following equation:

\[
\text{dst}[n] = \frac{1}{N-1} \sum_{k=0}^{N-1} \text{src}[k] \times \text{window}[k] \times \exp(j2\pi n k/N)
\]

where

\[
\begin{align*}
&k = 0, 1, ..., (N - 1) \\
n &= 0, 1, ..., (N - 1) \\
N &= 2^{\text{order}}
\end{align*}
\]

The signal FFTW/IFFTW functions can be categorized into four groups according to the `ScaleMode` in the function names in the following form:

- `mlib_Signal[FTTW|IFFTW]_ScaleMode_OutType_InType_OpMode()`
- `mlib_Signal[FTTW|IFFTW]_ScaleMode_DataType_OpMode()`

The scaling factors C1 and C2 used in the equations are defined as follows:

- For `ScaleMode = 1`, C1 = 1 and C2 = 2**order.
- For `ScaleMode = 2`, C1 = 2**order and C2 = 1.
- For `ScaleMode = 3`, C1 = C2 = 2**((order/2) when order is even, or C1 = 2**(order+1)/2) and C2 = 2**((order-1)/2) when order is odd.
- For `ScaleMode = 4`, C1 = 2**P and C2 = 2**Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

### Parameters
Each function takes some of the following arguments:

- `dstr` Destination signal array that contains the real parts.
- `dsti` Destination signal array that contains the imaginary parts.
- `srcr` Source signal array that contains the real parts.
- `srci` Source signal array that contains the imaginary parts.
Complex destination signal array. \texttt{dstc}[2*i] contains the real parts, and \texttt{dstc}[2*i+1] contains the imaginary parts.

Complex source signal array. \texttt{srcc}[2*i] contains the real parts, and \texttt{srcc}[2*i+1] contains the imaginary parts.

Source and destination signal array that contains the real parts.

Source and destination signal array that contains the imaginary parts.

Complex source and destination signal array. \texttt{srcdstc}[2*i] contains the real parts, and \texttt{srcdstc}[2*i+1] contains the imaginary parts.

Window coefficient array with \(2^\text{order}\) real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

Order of the transformation. The base-2 logarithm of the number of data samples.

Each function returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also \texttt{mlib_SignalFFTW_1(3MLIB)}, \texttt{mlib_SignalFFTW_2(3MLIB)}, \texttt{mlib_SignalFFTW_3(3MLIB)}, \texttt{mlib_SignalFFTW_4(3MLIB)}, \texttt{mlib_SignalIFFTW_1(3MLIB)}, \texttt{mlib_SignalIFFTW_3(3MLIB)}, \texttt{mlib_SignalIFFTW_4(3MLIB)}, attributes(5)
mlib_SignalIFFTW_3, mlib_SignalIFFTW_3_S16_S16_Mod, mlib_SignalIFFTW_3_S16C_S16C_Mod, mlib_SignalIFFTW_3_S16_S16C_Mod, mlib_SignalIFFTW_3_S16_Mod, mlib_SignalIFFTW_3_S16C_Mod, mlib_SignalIFFTW_3_F32_F32, mlib_SignalIFFTW_3_F32C_F32C, mlib_SignalIFFTW_3_F32_F32C, mlib_SignalIFFTW_3_F32C, mlib_SignalIFFTW_3_F32C

signalInverseFastFourierTransformwithwindowing (IFFTW)

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_SignalIFFTW_3_S16_S16_Mod(mlib_s16 *dstr, mlib_s16 *dsti, const mlib_s16 *srcr, const mlib_s16 *srci, const mlib_s16 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_S16C_S16C_Mod(mlib_s16 *dstc, const mlib_s16 *srcc, const mlib_s16 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_S16_S16C_Mod(mlib_s16 *dstr, const mlib_s16 *srcr, const mlib_s16 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_S16C_S16C_Mod(mlib_s16 *dstc, const mlib_s16 *srcc, const mlib_s16 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_S16_S16C_Mod(mlib_s16 *dstr, const mlib_s16 *srcr, const mlib_s16 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_S16C_S16C_Mod(mlib_s16 *dstc, const mlib_s16 *srcc, const mlib_s16 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_F32_F32(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *srcr, const mlib_f32 *srci, const mlib_f32 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_F32C_F32C(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_F32_C32C(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_F32C_S32C(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_F32C_S32C(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *window, mlib_s32 order);

mlib_status mlib_SignalIFFTW_3_F32_C32C(mlib_f32 *dstr, mlib_f32 *dsti, const mlib_f32 *window, mlib_s32 order);
Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

\[
\text{dst}[k] = \sum_{n=0}^{N-1} \text{src}[n] \times \text{window}[n] \times \exp(-j2\pi n k/N) \times C1
\]

and the IFFT W functions use the following equation:

\[
\text{dst}[n] = \sum_{k=0}^{N-1} \text{src}[k] \times \text{window}[k] \times \exp(j2\pi n k/N) \times C2
\]

where

k = 0, 1, ..., (N - 1)
n = 0, 1, ..., (N - 1)
N = 2**order

The scaling factors C1 and C2 used in the equations are defined as follows:

- For ScaleMode = 1, C1 = 1 and C2 = 2**order.
- For ScaleMode = 2, C1 = 2**order and C2 = 1.
- For ScaleMode = 3, C1 = C2 = 2**((order/2) when order is even, or C1 = 2**(order+1)/2) and C2 = 2**((order-1)/2) when order is odd.
- For ScaleMode = 4, C1 = 2**P and C2 = 2**Q, where P and Q are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**

Each function takes some of the following arguments:

- **dstr**: Destination signal array that contains the real parts.
- **dsti**: Destination signal array that contains the imaginary parts.
- **srcr**: Source signal array that contains the real parts.
- **srci**: Source signal array that contains the imaginary parts.
dstc Complex destination signal array. \( \text{dstc}[2*i] \) contains the real parts, and \( \text{dstc}[2*i+1] \) contains the imaginary parts.

src Complex source signal array. \( \text{src}[2*i] \) contains the real parts, and \( \text{src}[2*i+1] \) contains the imaginary parts.

srcdstr Source and destination signal array that contains the real parts.

srcdsti Source and destination signal array that contains the imaginary parts.

srcdstc Complex source and destination signal array. \( \text{srcdstc}[2*i] \) contains the real parts, and \( \text{srcdstc}[2*i+1] \) contains the imaginary parts.

window Window coefficient array with \( 2^{**\text{order}} \) real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.

order Order of the transformation. The base-2 logarithm of the number of data samples.

**Return Values** Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_SignalFFTW_1(3MLIB), mlib_SignalFFTW_2(3MLIB), mlib_SignalFFTW_3(3MLIB), mlib_SignalFFTW_4(3MLIB), mlib_SignalIFFTW_1(3MLIB), mlib_SignalIFFTW_2(3MLIB), mlib_SignalIFFTW_3(3MLIB), mlib_SignalIFFTW_4(3MLIB), attributes(5)
Each of the functions in this group performs Inverse Fast Fourier Transform with windowing (IFFTW).

The FFTW functions use the following equation:

\[ \text{dst}[k] = \frac{1}{N} \sum_{n=0}^{N-1} \text{src}[n] \times \text{window}[n] \times \exp(-j2\pi n k / N) \]

and the IFFTW functions use the following equation:

\[ \text{dst}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \text{src}[k] \times \text{window}[k] \times \exp(j2\pi n k / N) \]

where

- \( k = 0, 1, \ldots, (N - 1) \)
- \( n = 0, 1, \ldots, (N - 1) \)
- \( N = 2^{\text{order}} \)

The signal FFTW/IFFTW functions can be categorized into four groups according to the ScaleMode in the function names in the following form:

- \text{mlib\_Signal\[FFTW|IFFTW\]\_ScaleMode\_OutType\_InType\_OpMode()}
The scaling factors $C_1$ and $C_2$ used in the equations are defined as follows:

- For $\text{ScaleMode} = 1$, $C_1 = 1$ and $C_2 = 2^{*\text{order}}$.
- For $\text{ScaleMode} = 2$, $C_1 = 2^{*\text{order}}$ and $C_2 = 1$.
- For $\text{ScaleMode} = 3$, $C_1 = C_2 = 2^{*(\text{order}/2)}$ when $\text{order}$ is even, or $C_1 = 2^{*((\text{order}+1)/2)}$ and $C_2 = 2^{*((\text{order}-1)/2)}$ when $\text{order}$ is odd.
- For $\text{ScaleMode} = 4$, $C_1 = 2^{*P}$ and $C_2 = 2^{*Q}$, where $P$ and $Q$ are adaptive scaling factors and are generated by the functions.

For functions with only real parts for the source signal, the imaginary parts are assumed to be all zero. For functions with only real parts for the destination signal, the imaginary parts are discarded. The functions with only one data type in their names perform the operation in place.

**Parameters**

Each function takes some of the following arguments:

- $\text{dstr}$: Destination signal array that contains the real parts.
- $\text{dsti}$: Destination signal array that contains the imaginary parts.
- $\text{srcr}$: Source signal array that contains the real parts.
- $\text{srci}$: Source signal array that contains the imaginary parts.
- $\text{dstc}$: Complex destination signal array. $\text{dstc}[2*i]$ contains the real parts, and $\text{dstc}[2*i+1]$ contains the imaginary parts.
- $\text{srcc}$: Complex source signal array. $\text{srcc}[2*i]$ contains the real parts, and $\text{srcc}[2*i+1]$ contains the imaginary parts.
- $\text{srcdstr}$: Source and destination signal array that contains the real parts.
- $\text{srcdsti}$: Source and destination signal array that contains the imaginary parts.
- $\text{srcdstc}$: Complex source and destination signal array. $\text{srcdstc}[2*i]$ contains the real parts, and $\text{srcdstc}[2*i+1]$ contains the imaginary parts.
- $\text{window}$: Window coefficient array with $2^{*\text{order}}$ real elements. The window coefficients are in Q15 format for the S16 data type, or in float format for the F32 data type.
- $\text{order}$: Order of the transformation. The base-2 logarithm of the number of data samples.
- $\text{scale}$: Adaptive scaling factor.

**Return Values**

Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:
## mlib_SignalFFTW_4(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalFFTW_1(3MLIB), mlib_SignalFFTW_2(3MLIB), mlib_SignalFFTW_3(3MLIB), mlib_SignalFFTW_4(3MLIB), mlib_SignalIFFTW_1(3MLIB), mlib_SignalIFFTW_2(3MLIB), mlib_SignalIFFTW_3(3MLIB), attributes(5)
mlib_SignalIIR_Biquad_S16_S16_Sat, mlib_SignalIIR_Biquad_S16S_S16S_Sat,
mlib_SignalIIR_Biquad_F32_F32, mlib_SignalIIR_Biquad_F32S_F32S
- biquad Infinite Impulse Response (IIR) filtering

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalIIR_Biquad_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalIIR_Biquad_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalIIR_Biquad_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalIIR_Biquad_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, void *filter, mlib_s32 n);

Description Each of these functions applies a biquad IIR filter to a signal array.

\[
X = x(n) \quad n = 0, 1, ...
\]

\[
Z = z(n) \quad n = 0, 1, ...
\]

\[
N = \sum_{k=0}^{M} a_k x(n-k) + \sum_{k=1}^{M} b_k z(n-k) \quad n = 0, 1, ...
\]

\[
H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}
\]

The biquad IIR filter is represented by the following figure:
Each of the functions takes the following arguments:

- \textit{dst} Destination signal array.
- \textit{src} Source signal array.
- \textit{filter} Internal filter structure.
- \textit{n} Number of samples in the source signal array.

Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also \texttt{mlib\_SignalIIR\_Biquad\_S16\_S16\_Sat(3MLIB)}, \texttt{mlib\_SignalIIRFree\_Biquad\_S16\_S16(3MLIB)}, \texttt{mlib\_SignalIIRFree\_P4\_S16\_S16(3MLIB)}, \texttt{mlib\_SignalIIRInit\_Biquad\_S16\_S16(3MLIB)}, \texttt{mlib\_SignalIIRInit\_P4\_S16\_S16(3MLIB)}, attributes(5)
mlib_SignalIIRFree_Biquad_F32_F32, mlib_SignalIIRFree_Biquad_F32S_F32S – biquad Infinite Impulse Response (IIR) filtering

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalIIRFree_Biquad_F32_F32(void *filter);
void mlib_SignalIIRFree_Biquad_F32S_F32S(void *filter);

Description

Each of these functions releases the memory allocated for the internal filter structure.

Parameters

Each of the functions takes the following arguments:

filter Internal filter structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),
mlib_SignalIIR_P4_S16_S16_Sat(3MLIB), mlib_SignalIIRFree_P4_S16_S16(3MLIB),
mlib_SignalIIRInit_Biquad_S16_S16(3MLIB),
mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
mlib_SignalIIRFree_Biquad_S16_S16, mlib_SignalIIRFree_Biquad_S16S_S16S – biquad Infinite Impulse Response (IIR) filtering

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalIIRFree_Biquad_S16_S16(void *filter);
void mlib_SignalIIRFree_Biquad_S16S_S16S(void *filter);

Description

Each of these functions releases the memory allocated for the internal filter structure.

Parameters

Each of the functions takes the following arguments:

- filter Internal filter structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),
mlib_SignalIIR_P4_S16_S16_Sat(3MLIB), mlib_SignalIIRFree_P4_S16_S16(3MLIB),
mlib_SignalIIRInit_Biquad_S16_S16(3MLIB),
mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
Name: mlib_SignalIIRFree_P4_F32_F32, mlib_SignalIIRFree_P4_F32S_F32S – parallel Infinite Impulse Response (IIR) filtering

Synopsis:
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalIIRFree_P4_F32_F32(void *filter);
void mlib_SignalIIRFree_P4_F32S_F32S(void *filter);
```

Description: Each of these functions releases the memory allocated for the internal filter structure.

Parameters: Each of the functions takes the following arguments:
- `filter`: Internal filter structure.

Return Values: None.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB), mlib_SignalIIR_P4_S16_S16_Sat(3MLIB), mlib_SignalIIRFree_Biquad_S16_S16(3MLIB), mlib_SignalIIRInit_Biquad_S16_S16(3MLIB), mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
mlib_SignalIIRFree_P4_S16_S16(3MLIB)

Name  
mlib_SignalIIRFree_P4_S16_S16, mlib_SignalIIRFree_P4_S16S_S16S – parallel Infinite
Impulse Response (IIR) filtering

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalIIRFree_P4_S16_S16(void *filter);
void mlib_SignalIIRFree_P4_S16S_S16S(void *filter);

Description  
Each of these functions releases the memory allocated for the internal filter structure.

Parameters  
Each of the functions takes the following arguments:

  filter  Internal filter structure.

Return Values  
None.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),
mlib_SignalIIR_P4_S16_S16_Sat(3MLIB),
mlib_SignalIIRFree_Biquad_S16_S16(3MLIB),
mlib_SignalIIRInit_Biquad_S16_S16(3MLIB),
mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
mlib_SignalIIRInit_Biquad_F32_F32

**Name**
mlib_SignalIIRInit_Biquad_F32_F32, mlib_SignalIIRInit_Biquad_F32S_F32S – biquad Infinite Impulse Response (IIR) filtering

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalIIRInit_Biquad_F32_F32(void **filter, 
const mlib_f32 *flt);
mlib_status mlib_SignalIIRInit_Biquad_F32S_F32S(void **filter, 
const mlib_f32 *flt);

**Description**
Each of these functions allocates memory for the internal file structure and converts the filter coefficients into an internal representation.

**Parameters**
Each of the functions takes the following arguments:

- **filter** Internal filter structure.
- **flt** Array of five filter coefficients: a0, a1, a2, b1, and b2.

**Return Values**
Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),
mlib_SignalIIR_P4_S16_S16_Sat(3MLIB),
mlib_SignalIIRFree_Biquad_S16_S16(3MLIB),
mlib_SignalIIRFree_P4_S16_S16(3MLIB),mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
Infinite Impulse Response (IIR) filtering

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalIIRInit_Biquad_S16_S16(void **filter, const mlib_f32 *flt);
mlib_status mlib_SignalIIRInit_Biquad_S16S_S16S(void **filter, const mlib_f32 *flt);

Each of these functions allocates memory for the internal file structure and converts the filter coefficients into an internal representation.

Parameters
Each of the functions takes the following arguments:

filter Internal filter structure.
flt Array of five filter coefficients: a0, a1, a2, b1, and b2.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB), mlib_SignalIIR_P4_S16_S16_Sat(3MLIB), mlib_SignalIIRFree_Biquad_S16_S16(3MLIB), mlib_SignalIIRFree_P4_S16_S16(3MLIB), mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
mlib_SignalIIRInit_P4_F32_F32, mlib_SignalIIRInit_P4_F32S_F32S – parallel Infinite Impulse Response (IIR) filtering

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalIIRInit_P4_F32_F32(void **filter, const mlib_f32 flt);
mlib_status mlib_SignalIIRInit_P4_F32S_F32S(void **filter, const mlib_f32 flt);
```

Description
Each of these functions allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

Parameters
Each of the functions takes the following arguments:

- `filter` Internal filter structure.
- `flt` Array of nine filter coefficients: c, a00, a10, b10, b20, a01, a11, b11, and b21.

Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB), mlib_SignalIIR_P4_S16_S16_Sat(3MLIB), mlib_SignalIIRFree_Biquad_S16_S16(3MLIB), mlib_SignalIIRFree_P4_S16_S16(3MLIB), mlib_SignalIIRInit_Biquad_S16_S16(3MLIB), attributes(5)
mlib_SignalIIRInit_P4_S16_S16, mlib_SignalIIRInit_P4_S16S_S16S – parallel Infinite Impulse Response (IIR) filtering

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalIIRInit_P4_S16_S16(void **filter, const mlib_f32 flt);
mlib_status mlib_SignalIIRInit_P4_S16S_S16S(void **filter, const mlib_f32 flt);

Description  
Each of these functions allocates memory for the internal filter structure and converts the filter coefficients to an internal representation.

Parameters  
Each of the functions takes the following arguments:

- **filter**  
  Internal filter structure.

- **flt**  
  Array of nine filter coefficients: c, a00, a10, b10, b20, a01, a11, b11, and b21.

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),
mlib_SignalIIR_P4_S16_S16_Sat(3MLIB),
mlib_SignalIIRFree_Biquad_S16_S16(3MLIB),
mlib_SignalIIRFree_P4_S16_S16(3MLIB),
mlib_SignalIIRInit_Biquad_S16_S16(3MLIB), attributes(5)
Name  mlib_SignalIIR_P4_S16_S16_Sat, mlib_SignalIIR_P4_S16S_S16S_Sat, mlib_SignalIIR_P4_F32_F32, mlib_SignalIIR_P4_F32S_F32S – parallel Infinite Impulse Response (IIR) filtering

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_SignalIIR_P4_S16_S16_Sat(mlib_s16 *dst, 
    const mlib_s16 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalIIR_P4_S16S_S16S_Sat(mlib_s16 *dst, 
    const mlib_s16 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalIIR_P4_F32_F32(mlib_f32 *dst, 
    const mlib_f32 *src, void *filter, mlib_s32 n);
mlib_status mlib_SignalIIR_P4_F32S_F32S(mlib_f32 *dst, 
    const mlib_f32 *src, void *filter, mlib_s32 n);
```

Description  Each of these functions applies a fourth order parallel IIR filter to one signal packet and updates the filter state.

\[
X = x(n) \quad n = 0, 1, ...
\]

\[
Z = z(n) \quad n = 0, 1, ...
\]

\[
N = \sum_{k=0}^{M} a_k x(n-k) + \sum_{k=1}^{M} b_k z(n-k) \quad n = 0, 1, ...
\]

\[
H(z) = C + \sum_{k=0}^{1} \frac{(a_{2k} + a_{1k}z^{-1})}{(1 + b_{1k}z^{-1} + b_{2k}z^{-2})}
\]

The fourth order parallel IIR filter is represented by the following figure:
Each of the functions takes the following arguments:

- `dst` -- Destination signal array.
- `src` -- Source signal array.
- `filter` -- Internal filter structure.
- `n` -- Number of signal samples.

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_SignalIIR_Biquad_S16_S16_Sat(3MLIB),
mlib_SignalIIRFree_Biquad_S16_S16(3MLIB),
mlib_SignalIIRFree_P4_S16_S16(3MLIB),
mlib_SignalIIRInit_Biquad_S16_S16(3MLIB),
mlib_SignalIIRInit_P4_S16_S16(3MLIB), attributes(5)
The `mlib_SignalIMDCT_D64()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

Parameters


Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_SignalIMDCT_F32(3MLIB)`, `mlib_SignalIMDCTSplit_D64(3MLIB)`, `mlib_SignalIMDCTSplit_F32(3MLIB)`, `attributes(5)`
The `mlib_SignalIMDCT_F32()` function performs the inverse modified discrete cosine transformation in Dolby’s AC-3 digital audio standard.

### Parameters
The function takes the following arguments:

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_SignalIMDCT_D64(3MLIB)`, `mlib_SignalIMDCTSplit_D64(3MLIB)`,
- `mlib_SignalIMDCTSplit_F32(3MLIB)`, `attributes(5)`
The `mlib_SignalIMDCTSplit_D64()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

**Parameters**
The function takes the following arguments:
- `data` Pointer to the data array. `data[4*i]` contains the real parts of the first array, `data[4*i+1]` contains the real parts of the second array, `data[4*i+2]` contains the imaginary parts of the first array, and `data[4*i+3]` contains the imaginary parts of the second array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_SignalIMDCT_D64(3MLIB), mlib_SignalIMDCT_F32(3MLIB), mlib_SignalIMDCTSplit_F32(3MLIB), attributes(5)`
The `mlib_SignalIMDCTSplit_F32()` function performs the inverse modified discrete cosine transformation in Dolby's AC-3 digital audio standard.

The function takes the following arguments:

- `data` Pointer to the data array. `data[4*i]` contains the real parts of the first array, `data[4*i+1]` contains the real parts of the second array, `data[4*i+2]` contains the imaginary parts of the first array, and `data[4*i+3]` contains the imaginary parts of the second array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_SignalIMDCT_D64(3MLIB), mlib_SignalIMDCT_F32(3MLIB), mlib_SignalIMDCTSplit_D64(3MLIB), attributes(5)`
Each of these functions performs hard limiting.

For monaural signals, the following equation is used:

\[
\begin{align*}
\text{dst}[i] &= \text{low}[0] & \text{if } \text{src}[i] < \text{low}[0] \\
\text{dst}[i] &= \text{src}[i] & \text{if } \text{low}[0] \leq \text{src}[i] < \text{high}[0] \\
\text{dst}[i] &= \text{high}[0] & \text{if } \text{src}[i] \geq \text{high}[0]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

For stereo signals, the following equation is used:

\[
\begin{align*}
\text{dst}[2*i] &= \text{low}[0] & \text{if } \text{src}[2*i] < \text{low}[0] \\
\text{dst}[2*i] &= \text{src}[2*i] & \text{if } \text{low}[0] \leq \text{src}[2*i] < \text{high}[0] \\
\text{dst}[2*i] &= \text{high}[0] & \text{if } \text{src}[2*i] \geq \text{high}[0] \\
\text{dst}[2*i+1] &= \text{low}[1] & \text{if } \text{src}[2*i+1] < \text{low}[1]
\end{align*}
\]
$$dst[2*i+1] = high[1] \text{ if src[2*i+1] } \geq high[1]$$

where $$i = 0, 1, \ldots, (n - 1).$$

**Parameters** Each of the functions takes some of the following arguments:

- **dst**: Destination signal array.
- **src**: Source signal array.
- **srcdst**: Source and destination signal array.
- **low**: Lower input limit. In the stereo version, low[0] contains the lower limit for channel 0, and low[1] contains the lower limit for channel 1.
- **high**: Upper input limit. In the stereo version, high[0] contains the upper limit for channel 0, and high[1] contains the upper limit for channel 1.
- **n**: Number of samples in the source signal array.

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)
The `mlib_SignalLinear2ADPCM2Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 2-bit ADPCM to 16-bit linear PCM to G.723 or G.726 16kbps format.

The function takes the following arguments:

- `adpcm` ADPCM code array.
- `pcm` Linear PCM sample array.
- `state` Internal structure of the codec.
- `n` Number of samples in the source array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalADPCM2Bits2Linear(3MLIB)`
- `mlib_SignalADPCM3Bits2Linear(3MLIB)`
- `mlib_SignalADPCM4Bits2Linear(3MLIB)`
- `mlib_SignalADPCM5Bits2Linear(3MLIB)`
- `mlib_SignalADPCMFree(3MLIB)`
- `mlib_SignalADPCMInit(3MLIB)`
- `mlib_SignalLinear2ADPCM3Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM4Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM5Bits(3MLIB)`
- `attributes(5)`
# mlib_SignalLinear2ADPCM3Bits

## Name
mlib_SignalLinear2ADPCM3Bits – adaptive differential pulse code modulation (ADPCM)

## Synopsis
```c
#include <mlib.h>

mlib_status mlib_SignalLinear2ADPCM3Bits(mlib_u8 *adpcm,
    const mlib_s16 *pcm, void *state, mlib_s32 n);
```

## Description
The `mlib_SignalLinear2ADPCM3Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 16-bit linear PCM to G.723 or G.726 24kbps 3-bit ADPCM format.

## Parameters
The function takes the following arguments:
- `adpcm` : ADPCM code array.
- `pcm` : Linear PCM sample array.
- `state` : Internal structure of the codec.
- `n` : Number of samples in the source array.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_SignalADPCM2Bits2Linear(3MLIB)`, `mlib_SignalADPCM3Bits2Linear(3MLIB)`,
- `mlib_SignalADPCM4Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`,
- `mlib_SignalADPCMFree(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`,
- `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM4Bits(3MLIB)`,
- `mlib_SignalLinear2ADPCM5Bits(3MLIB)`, attributes(5)
**mlib_SignalLinear2ADPCM4Bits** - adaptive differential pulse code modulation (ADPCM)

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLinear2ADPCM4Bits(mlib_u8 *adpcm,
                                          const mlib_s16 *pcm, void *state, mlib_s32 n);
```

**Description**
The `mlib_SignalLinear2ADPCM4Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 16-bit linear PCM to G.721 or G.726 32kbps 4-bit ADPCM format.

**Parameters**
The function takes the following arguments:
- `adpcm` : ADPCM code array.
- `pcm` : Linear PCM sample array.
- `state` : Internal structure of the codec.
- `n` : Number of samples in the source array.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_SignalADPCM2Bits2Linear(3MLIB)`, `mlib_SignalADPCM3Bits2Linear(3MLIB)`, `mlib_SignalADPCM4Bits2Linear(3MLIB)`, `mlib_SignalADPCM5Bits2Linear(3MLIB)`, `mlib_SignalADPCMFree(3MLIB)`, `mlib_SignalADPCMInit(3MLIB)`, `mlib_SignalLinear2ADPCM2Bits(3MLIB)`, `mlib_SignalLinear2ADPCM3Bits(3MLIB)`, `mlib_SignalLinear2ADPCM5Bits(3MLIB)`, `attributes(5)`
The `mlib_SignalLinear2ADPCM5Bits()` function performs adaptive differential pulse code modulation (ADPCM) in compliance with the ITU (former CCITT) G.721, G.723, and G.726 specifications. It converts data from 16-bit linear PCM to G.723 or G.726 40kbps 5-bit ADPCM format.

The function takes the following arguments:

- `adpcm`: ADPCM code array.
- `pcm`: Linear PCM sample array.
- `state`: Internal structure of the codec.
- `n`: Number of samples in the source array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_SignalADPCM2Bits2Linear(3MLIB)`
- `mlib_SignalADPCM3Bits2Linear(3MLIB)`
- `mlib_SignalADPCM4Bits2Linear(3MLIB)`
- `mlib_SignalADPCM5Bits2Linear(3MLIB)`
- `mlib_SignalADPCMFree(3MLIB)`
- `mlib_SignalADPCMInit(3MLIB)`
- `mlib_SignalLinear2ADPCM2Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM3Bits(3MLIB)`
- `mlib_SignalLinear2ADPCM4Bits(3MLIB)`, `attributes(5)`
mlib_SignalLinear2ALaw – ITU G.711 m-law and A-law compression and decompression

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLinear2ALaw(mlib_u8 *acode,
        const mlib_s16 *pcm, mlib_s32 n);
```

Description

The `mlib_SignalLinear2ALaw()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

Parameters

The function takes the following arguments:

- `acode` A-law code array.
- `pcm` Linear PCM sample array.
- `n` Number of samples in the source array.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_SignalALaw2Linear(3MLIB)`, `mlib_SignalALaw2uLaw(3MLIB)`,
`mlib_SignalLinear2uLaw(3MLIB)`, `mlib_SignaluLaw2ALaw(3MLIB)`,
`mlib_SignaluLaw2Linear(3MLIB)`
mlib_SignalLinear2uLaw – ITU G.711 m-law and A-law compression and decompression

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLinear2uLaw(mlib_u8 *ucode, const mlib_s16 *pcm, mlib_s32 n);

Description

The mlib_SignalLinear2uLaw() function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (Former CCITT) G.711 specification.

Parameters

The function takes the following arguments:

- **ucode**: m-law code array.
- **pcm**: Linear PCM sample array.
- **n**: Number of samples in the source array.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalALaw2Linear(3MLIB), mlib_SignalALaw2uLaw(3MLIB), mlib_SignalLinear2ALaw(3MLIB), mlib_SignaluLaw2ALaw(3MLIB), mlib_SignaluLaw2Linear(3MLIB), attributes(5)
Name  mlib_SignalLMSFilter, mlib_SignalLMSFilterInit_S16_S16,
      mlib_SignalLMSFilterInit_S16S_S16S, mlib_SignalLMSFilterInit_F32_F32,
      mlib_SignalLMSFilterInit_F32S_F32S, mlib_SignalLMSFilter_S16_S16_Sat,
      mlib_SignalLMSFilter_S16S_S16S_Sat, mlib_SignalLMSFilter_S16_S16_Sat,
      mlib_SignalLMSFilter_S16S_S16S_Sat, mlib_SignalLMSFilterNonAdapt_S16_S16_Sat,
      mlib_SignalLMSFilterNonAdapt_S16S_S16S_Sat,
      mlib_SignalLMSFilterNonAdapt_F32_S32, mlib_SignalLMSFilterNonAdapt_F32S_F32S,
      mlib_SignalLMSFilterFree_S16_S16, mlib_SignalLMSFilterFree_S16S_S16S_Sat,
      mlib_SignalLMSFilterFree_F32_F32S, mlib_SignalLMSFilterFree_F32S_F32S – least mean
      square (LMS) adaptive filtering

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLMSFilterInit_S16_S16(void **filter,
      const mlib_f32 *flt,
      mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalLMSFilterInit_S16S_S16S(void **filter,
      const mlib_f32 *flt,
      mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalLMSFilterInit_F32_F32(void **filter,
      const mlib_f32 *flt,
      mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalLMSFilterInit_F32S_F32S(void **filter,
      const mlib_f32 *flt,
      mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalLMSFilter_S16_S16_Sat(mlib_s16 *dst,
      const mlib_s16 *src,
      const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalLMSFilter_S16S_S16S_Sat(mlib_s16 *dst,
      const mlib_s16 *src,
      const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalLMSFilter_F32_F32(mlib_f32 *dst,
      const mlib_f32 *src,
      const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalLMSFilterF32S_F32S(mlib_f32 *dst,
      const mlib_f32 *src,
      const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalLMSFilterNonAdapt_S16_S16_Sat(mlib_s16 *dst,
      const mlib_s16 *src, const mlib_s16 *ref,
      void *filter, mlib_s32 n);
The basic LMS adaptive algorithm is summarized as follows:

1. Initialize the weights $W_k(i)$, $i = 0, 1, \ldots, \text{tap} - 1$.
2. Initialize previous source elements $X_o(i)$, $i = 0, 1, \ldots, \text{tap} - 1$.
3. Read $X_k(t)$ from $\text{src}$ and $Y_k(t)$ from $\text{ref}$, $t = 0, 1, \ldots, n - 1$.
4. Compute filter output: $n_k = \sum(W_k(i) \cdot X_k(t - i))$, $i = 0, 1, \ldots, \text{tap} - 1$. If $i > t$, use previous source elements stored in the $X_o$ vector.
5. Store filter output: $\text{dst}[t] = n_k$.
6. Compute the error estimate: $E_k = Y_k - n_k$.
7. Compute factor $B E_0 = 2 \cdot \beta \cdot E_k$.
8. Update filter weights: $W_k(i) += B E_0 \cdot X_k(t - i)$, $i = 0, 1, \ldots, \text{tap} - 1$. If $i > t$, use previous source elements stored in $X_o$ vector.
9. Next $t$, go to step 3.
10. Store $N$ ending source elements in previous source elements vector $X_o$: if $N > n$, $N = n$; else $N = \text{tap}$.

The functions assume that the input signal has a power maximum equal to 1. If it is not, $\beta$ should be divided by power maximum. Power maximum is calculated according to the following formula:

$$\text{Power\_max} = \max \left\{ \frac{\sum_{n=0}^{\text{flt\_len}} \text{signal}(n + k)^2}{n-k=0} \right\}$$

It is necessary to consider the maximum of power maxima of both components as the stereo signal’s power maximum.
Each of the FilterInit functions allocates memory for the internal filter structure and converts the parameters into the internal representation.

Each of the Filter functions applies the LMS adaptive filter on one signal packet and updates the filter states.

Each of the FilterNoAdapt functions applies the LMS filter on one signal packet and updates the filter states but without changing the filter weights.

Each of the FilterFree functions releases the memory allocated for the internal filter structure.

**Parameters** Each of the functions takes some of the following arguments:

- filter Internal filter structure.
- flt Filter coefficient array.
- tap Taps of the filter.
- beta Error weighting factor. $0 < \beta < 1$.
- dst Destination signal array.
- src Source signal array.
- ref Reference or “desired” signal array.
- n Number of samples in the source signal array.

**Return Values** Each of the FilterInit, Filter and FilterNoAdapt functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE. The FilterFree functions don’t return anything.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_SignalNLMSFilter(3MLIB), attributes(5)
The `mlib_SignalLPC2Cepstral_F32()` function converts linear prediction coefficients to cepstral coefficients.

The cepstral coefficients are the coefficients of the Fourier transform representation of the log magnitude spectrum.

The LPC cepstral coefficients can be derived recursively from the LPC coefficients as following.

\[
c(0) = \log(G)
\]

\[
c(m) = a(m) + \sum_{k=1}^{m-1} c(k) * a(m-k), 1 \leq m \leq M
\]

\[
c(m) = \sum_{k=1}^{m-1} c(k) * a(m-k), m > M
\]


The function takes the following arguments:

- `cepst` The cepstral coefficients.
- `lpc` The linear prediction coefficients.
- `gain` The gain of the LPC model.
- `length` The length of the cepstral coefficients.
- `order` The order of the linear prediction filter.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Parameters**

**Return Values**

**Attributes**

See attributes(5) for descriptions of the following attributes:
### mlib_SignalLPC2Cepstral_F32(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalLPC2Cepstral_S16(3MLIB), mlib_SignalLPC2Cepstral_S16_Adp(3MLIB), mlib_SignalLPC2Cepstral_F32(3MLIB), attributes(5)
The `mlib_SignalLPC2Cepstral_S16()` function converts linear prediction coefficients to cepstral coefficients. The user supplied scaling factor, `cscale`, will be used and the output will be saturated if necessary.

The cepstral coefficients are the coefficients of the Fourier transform representation of the log magnitude spectrum.

The LPC cepstral coefficients can be derived recursively from the LPC coefficients as following.

\[
    c(0) = \log(G)
\]

\[
    c(m) = a(m) + \sum_{k=1}^{m-1} c(k) \cdot a(m-k), \quad 1 \leq m \leq M
\]

\[
    c(m) = \sum_{k=1}^{m-1} c(k) \cdot a(m-k), \quad m > M
\]


### Parameters

The function takes the following arguments:

- `cepst` The cepstral coefficients.
- `cscale` The scaling factor of the cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- `lpc` The linear prediction coefficients.
- `lscale` The scaling factor of the linear prediction coefficients, where `actual_data = input_data * 2**(-scaling_factor)`.
- `gain` The gain of the LPC model.
- `gscale` The scaling factor of the gain of the LPC model, where `actual_data = input_data * 2**(-scaling_factor)`.
- `length` The length of the cepstral coefficients.
order  The order of the linear prediction filter.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPC2Cepstral_S16(3MLIB), mlib_SignalLPC2Cepstral_S16_Adp(3MLIB), mlib_SignalLPC2Cepstral_F32(3MLIB), attributes(5)
The `mlib_SignalLPC2Cepstral_S16_Adp()` function converts linear prediction coefficients to cepstral coefficients. The scaling factor of the output data, `cscale`, will be calculated based on the actual data.

The cepstral coefficients are the coefficients of the Fourier transform representation of the log magnitude spectrum.

The LPC cepstral coefficients can be derived recursively from the LPC coefficients as following.

\[
c(0) = \log(G) \\
\]

\[
c(m) = a(m) + \sum_{k=1}^{m-1} \frac{k}{m} c(k) * a(m-k), \ 1 \leq m \leq M \\
\]

\[
c(m) = \sum_{k=1}^{m-1} \frac{k}{m} c(k) * a(m-k), m > M \\
\]


**Parameters**

The function takes the following arguments:

- `cepst`  The cepstral coefficients.
- `cscale` The scaling factor of the cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.  
- `lpc`  The linear prediction coefficients.
- `lscale` The scaling factor of the linear prediction coefficients, where `actual_data = input_data * 2**(-scaling_factor)`.  
- `gain`  The gain of the LPC model.
- `gscale` The scaling factor of the gain of the LPC model, where `actual_data = input_data * 2**(-scaling_factor)`.  
- `length` The length of the cepstral coefficients.
order  The order of the linear prediction filter.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPC2Cepstral_S16(3MLIB), mlib_SignalLPC2Cepstral_S16_Adp(3MLIB), mlib_SignalLPC2Cepstral_F32(3MLIB), attributes(5)
The `mlib_SignalLPC2LSP_F32()` function converts linear prediction coefficients to line spectral pair coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

\[
P(z) = A(z) + z^{-1} A(z)
\]

\[
Q(z) = A(z) - z^{-1} A(z)
\]

where \( A(z) \) is the inverse filter

\[
A(z) = 1 - \sum_{i=1}^{M} a(i) z^{-i}
\]

Note that since \( P(z) \) is symmetric and \( Q(z) \) is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other. \( P(z) \) has a root at \( z = -1 \) (\( w = \pi \)) and \( Q(z) \) has a root at \( z = 1 \) (\( w = 0 \)).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

\[
q = \cos(w)
\]

where \( q \) is the LPS and \( w \) is the LPF.


**Parameters**

- `lsp` The line spectral pair coefficients.
- `lpc` The linear prediction coefficients.
- `order` The order of the linear prediction filter.
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  `mlib_SignalLSP2LPC_F32(3MLIB)`, attributes(5)
The `mlib_SignalLPC2LSP_S16()` function converts linear prediction coefficients to line spectral pair coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

\[
\begin{align*}
P(z) &= A(z) + z^{-1} A(z) \\
Q(z) &= A(z) - z^{-1} A(z)
\end{align*}
\]

where \( A(z) \) is the inverse filter

\[
A(z) = \frac{1}{1 - \sum_{i=1}^{M-1} a(i) z^{-i}}
\]

Note that since \( P(z) \) is symmetric and \( Q(z) \) is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other. \( P(z) \) has a root at \( z = -1 \) \((w = \pi)\) and \( Q(z) \) has a root at \( z = 1 \) \((w = 0)\).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

\[
q = \cos(w)
\]

where \( q \) is the LPS and \( w \) is the LPF.


**Parameters**

- **lsp** The line spectral pair coefficients in Q15 format.
- **lpc** The linear prediction coefficients.
- **lscale** The scaling factor of the linear prediction coefficients, where \(\text{actual\_data} = \text{input\_data} \times 2^{(-\text{scaling\_factor})} \).
- **order** The order of the linear prediction filter.

**Return Values**

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.
**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_SignalLSP2LPC_S16(3MLIB), attributes(5)
The `mlib_SignalLPCAutoCorrel_F32()` function performs linear predictive coding with autocorrelation method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past \( M \) samples.

\[
M \quad s(n) = \sum_{i=1}^{M} a(i) \cdot s(n-i) + G \cdot u(n)
\]

where \( s(*) \) is the speech signal, \( u(*) \) is the excitation signal, and \( G \) is the gain constants, \( M \) is the order of the linear prediction filter. Given \( s(*) \), the goal is to find a set of coefficient \( a(*) \) that minimizes the prediction error \( e(*) \).

\[
M \quad e(n) = s(n) - \sum_{i=1}^{M} a(i) \cdot s(n-i)
\]

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

\[
M \quad \sum_{i=1}^{M} a(i) \cdot r(|i-k|) = r(k), \quad k=1,\ldots,M
\]

where

\[
N-k-1 \quad r(k) = \sum_{j=0}^{N-k-1} s(j) \cdot s(j+k)
\]

are the autocorrelation coefficients of \( s(*) \), \( N \) is the length of the input speech vector. \( r(0) \) is the energy of the speech signal.

Note that the autocorrelation matrix \( R \) is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

The function takes the following arguments:

- `coeff`  The linear prediction coefficients.
- `signal`  The input signal vector.
- `state`   Pointer to the internal state structure.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: `mlib_SignalLPCAutoCorrelInit_F32(3MLIB)`,
          `mlib_SignalLPCAutoCorrelGetEnergy_F32(3MLIB)`,
          `mlib_SignalLPCAutoCorrelGetPARCOR_F32(3MLIB)`,
          `mlib_SignalLPCAutoCorrelFree_F32(3MLIB)`, attributes(5)
mlib_SignalLPCAutoCorrelFree_S16, mlib_SignalLPCAutoCorrelFree_F32 – clean up for autocorrelation method

Synopsis

cc [ flag...] file... -lmlib [ library...] 
#include <mlib.h>

void mlib_SignalLPCAutoCorrelFree_S16(void *state);
void mlib_SignalLPCAutoCorrelFree_F32(void *state);

Description

Each of these functions frees the internal state structure for autocorrelation method of linear predictive coding (LPC).

This function cleans up the internal state structure and releases all memory buffers.

Parameters

Each of the functions takes the following arguments:

state Pointer to the internal state structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalLPCAutoCorrelInit_S16(3MLIB),
mlib_SignalLPCAutoCorrelInit_F32(3MLIB),
mlib_SignalLPCAutoCorrel_S16(3MLIB),
mlib_SignalLPCAutoCorrel_S16_Adp(3MLIB),
mlib_SignalLPCAutoCorrel_F32(3MLIB),
mlib_SignalLPCAutoCorrelGetEnergy_S16(3MLIB),
mlib_SignalLPCAutoCorrelGetEnergy_F32(3MLIB),
mlib_SignalLPCAutoCorrelGetPARCOR_S16(3MLIB),
mlib_SignalLPCAutoCorrelGetPARCOR_F32(3MLIB), attributes(5)
The `mlib_SignalLPCAutoCorrelGetEnergy_F32()` function returns the energy of the input signal.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past \( M \) samples.

\[
M \quad s(n) = \sum_{i=1}^{M} a(i) \times s(n-i) + G \times u(n)
\]

where \( s(*) \) is the speech signal, \( u(*) \) is the excitation signal, and \( G \) is the gain constants, \( M \) is the order of the linear prediction filter. Given \( s(*) \), the goal is to find a set of coefficient \( a(*) \) that minimizes the prediction error \( e(*) \).

\[
M \quad e(n) = s(n) - \sum_{i=1}^{M} a(i) \times s(n-i)
\]

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

\[
M \quad \sum_{i=1}^{M} a(i) \times r(|i-k|) = r(k), \quad k=1,\ldots,M
\]

where

\[
N-k-1 \quad r(k) = \sum_{j=0}^{N-k-1} s(j) \times s(j+k)
\]

are the autocorrelation coefficients of \( s(*) \), \( N \) is the length of the input speech vector. \( r(0) \) is the energy of the speech signal.

Note that the autocorrelation matrix \( R \) is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

The function takes the following arguments:

- **energy**: The energy of the input signal.
- **state**: Pointer to the internal state structure.

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- mlib_SignalLPCAutoCorrelInit_F32(3MLIB)
- mlib_SignalLPCAutoCorrel_F32(3MLIB)
- mlib_SignalLPCAutoCorrelGetPARCOR_F32(3MLIB)
- mlib_SignalLPCAutoCorrelFree_F32(3MLIB)
- attributes(5)
mlib_SignalLPCAutoCorrelGetEnergy_S16, mlib_SignalLPCAutoCorrelGetEnergy_S16_Adp – return the energy of the input signal

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLPCAutoCorrelGetEnergy_S16(
    mlib_s16 *energy, mlib_s32 escale, void *state);

mlib_status mlib_SignalLPCAutoCorrelGetEnergy_S16_Adp(
    mlib_s16 *energy, mlib_s32 *escale, void *state);

Description

Each of the functions returns the energy of the input signal.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past $M$ samples.

$$ s(n) = \sum_{i=1}^{M} a(i) \cdot s(n-i) + G \cdot u(n) $$

where $s(*)$ is the speech signal, $u(*)$ is the excitation signal, and $G$ is the gain constants, $M$ is the order of the linear prediction filter. Given $s(*)$, the goal is to find a set of coefficient $a(*)$ that minimizes the prediction error $e(*)$.

$$ e(n) = s(n) - \sum_{i=1}^{M} a(i) \cdot s(n-i) $$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$ \sum_{i=1}^{M} a(i) \cdot r(|i-k|) = r(k), k=1,...,M $$

where

$$ r(k) = \sum_{j=0}^{N-k-1} s(j) \cdot s(j+k) $$

are the autocorrelation coefficients of $s(*)$, $N$ is the length of the input speech vector. $r(0)$ is the energy of the speech signal.

Note that the autocorrelation matrix $R$ is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

Note for functions with adaptive scaling (with _Adp postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without _Adp postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters** Each function takes the following arguments:

- **energy** The energy of the input signal.
- **escale** The scaling factor of the energy, where \( \text{actual} = \text{output_data} \times 2^{(-\text{scaling_factor})} \).
- **state** Pointer to the internal state structure.

**Return Values** Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_SignalLPCAutoCorrelInit_S16(3MLIB)`, `mlib_SignalLPCAutoCorrel_S16(3MLIB)`, `mlib_SignalLPCAutoCorrelGetPARCOR_S16(3MLIB)`, `mlib_SignalLPCAutoCorrelFree_S16(3MLIB)`, `attributes(5)`
The \texttt{mlib_SignalLPCAutoCorrelGetPARCOR\_F32()} function returns the partial correlation (PARCOR) coefficients.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past $M$ samples.

$$M \quad s(n) = \sum_{i=1}^{M} a(i) * s(n-i) + G * u(n)$$

where $s(\ast)$ is the speech signal, $u(\ast)$ is the excitation signal, and $G$ is the gain constants, $M$ is the order of the linear prediction filter. Given $s(\ast)$, the goal is to find a set of coefficient $a(\ast)$ that minimizes the prediction error $e(\ast)$.

$$M \quad e(n) = s(n) - \sum_{i=1}^{M} a(i) * s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$M \quad \sum_{i=1}^{M} a(i) * r(|i-k|) = r(k), \quad k=1,...,M$$

where

$$N-k-1 \quad r(k) = \sum_{j=0}^{N-k-1} s(j) * s(j+k)$$

are the autocorrelation coefficients of $s(\ast)$, $N$ is the length of the input speech vector. $r(0)$ is the energy of the speech signal.

Note that the autocorrelation matrix $R$ is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

The function takes the following arguments:

- `parcor` The partial correlation (PARCOR) coefficients.
- `state` Pointer to the internal state structure.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_SignalLPCAutoCorrelInit_F32(3MLIB), mlib_SignalLPCAutoCorrel_F32(3MLIB), mlib_SignalLPCAutoCorrelGetEnergy_F32(3MLIB), mlib_SignalLPCAutoCorrelFree_F32(3MLIB), attributes(5)`
Each of the functions returns the partial correlation (PARCOR) coefficients.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past $M$ samples.

$$M \quad s(n) = \sum_{i=1}^{M} a(i) \ast s(n-i) + G \ast u(n)$$

where $s(*)$ is the speech signal, $u(*)$ is the excitation signal, and $G$ is the gain constants, $M$ is the order of the linear prediction filter. Given $s(*)$, the goal is to find a set of coefficient $a(*)$ that minimizes the prediction error $e(*)$.

$$M \quad e(n) = s(n) - \sum_{i=1}^{M} a(i) \ast s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$M \quad \sum_{i=1}^{M} a(i) \ast r(|i-k|) = r(k), \quad k=1,\ldots,M$$

where

$$N-k-1 \quad r(k) = \sum_{j=0}^{N-k-1} s(j) \ast s(j+k)$$

are the autocorrelation coefficients of $s(*)$, $N$ is the length of the input speech vector. $r(0)$ is the energy of the speech signal.

Note that the autocorrelation matrix $R$ is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

Note for functions with adaptive scaling (with _Adp postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without _Adp postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters**

Each function takes the following arguments:

- `parcor`: The partial correlation (PARCOR) coefficients.
- `pscale`: The scaling factor of the partial correlation (PARCOR) coefficients, where \( \text{actual} = \text{output} \times 2^{(-\text{scaling} \_\text{factor})} \).
- `state`: Pointer to the internal state structure.

**Return Values**

Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_SignalLPCAutoCorrelInit_S16(3MLIB)`, `mlib_SignalLPCAutoCorrel_S16(3MLIB)`, `mlib_SignalLPCAutoCorrelGetEnergy_S16(3MLIB)`, `mlib_SignalLPCAutoCorrelFree_S16(3MLIB)`, attributes(5)
Name  mlib_SignalLPCAutoCorrelInit_S16, mlib_SignalLPCAutoCorrelInit_F32 – initialization for autocorrelation method of linear predictive coding

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLPCAutoCorrelInit_S16(void *state,
   mlib_s32 length, mlib_s32 order);
mlib_status mlib_SignalLPCAutoCorrelInit_F32(void *state,
   mlib_s32 length, mlib_s32 order);

Description  Each function initializes the internal state structure for autocorrelation method of linear predictive coding (LPC).

The init function performs internal state structure allocation and global initialization. Per LPC function call initialization is done in LPC function, so the same internal state structure can be reused for multiple LPC function calls.

Parameters  Each function takes the following arguments:
state     Pointer to the internal state structure.
length    The length of the input signal vector.
order     The order of the linear prediction filter.

Return Values  Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPCAutoCorrel_S16(3MLIB),
mlib_SignalLPCAutoCorrelGetEnergy_S16(3MLIB),
mlib_SignalLPCAutoCorrelGetPARCOR_S16(3MLIB),
mlib_SignalLPCAutoCorrelFree_S16(3MLIB), attributes(5)
Each function performs linear predictive coding with autocorrelation method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past $M$ samples.

$$s(n) = \sum_{i=1}^{M} a(i) \cdot s(n-i) + G \cdot u(n)$$

where $s(*)$ is the speech signal, $u(*)$ is the excitation signal, and $G$ is the gain constants, $M$ is the order of the linear prediction filter. Given $s(*)$, the goal is to find a set of coefficient $a(*)$ that minimizes the prediction error $e(*)$.

$$e(n) = s(n) - \sum_{i=1}^{M} a(i) \cdot s(n-i)$$

In autocorrelation method, the coefficients can be obtained by solving following set of linear equations.

$$\sum_{i=1}^{M} a(i) \cdot r(|i-k|) = r(k), \ k=1,...,M$$

where

$$r(k) = \sum_{j=0}^{N-k-1} s(j) \cdot s(j+k)$$

are the autocorrelation coefficients of $s(*)$, $N$ is the length of the input speech vector. $r(0)$ is the energy of the speech signal.

Note that the autocorrelation matrix $R$ is a Toeplitz matrix (symmetric with all diagonal elements equal), and the equations can be solved efficiently with Levinson-Durbin algorithm.

Note for functions with adaptive scaling (with \_Adp postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without \_Adp postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

Parameters Each function takes the following arguments:

- **coeff** The linear prediction coefficients.
- **cscale** The scaling factor of the linear prediction coefficients, where actual\_data = output\_data * 2**(-scaling\_factor).
- **signal** The input signal vector with samples in Q15 format.
- **state** Pointer to the internal state structure.

Return Values Each function returns MLIB\_SUCCESS if successful. Otherwise it returns MLIB\_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib\_SignalLPACorrelInit\_S16(3MLIB), mlib\_SignalLPACorrelGetEnergy\_S16(3MLIB), mlib\_SignalLPACorrelGetPARCOR\_S16(3MLIB), mlib\_SignalLPACorrelFree\_S16(3MLIB), attributes(5)
The `mlib_SignalLPCCovariance_F32()` function performs linear predictive coding with covariance method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past \( M \) samples.

\[
M \quad s(n) = \sum_{i=1}^{M} a(i) \cdot s(n-i) + G \cdot u(n)
\]

where \( s(*) \) is the speech signal, \( u(*) \) is the excitation signal, and \( G \) is the gain constants, \( M \) is the order of the linear prediction filter. Given \( s(*) \), the goal is to find a set of coefficient \( a(*) \) that minimizes the prediction error \( e(*) \).

\[
M \quad e(n) = s(n) - \sum_{i=1}^{M} a(i) \cdot s(n-i)
\]

In covariance method, the coefficients can be obtained by solving following set of linear equations.

\[
M \quad \sum_{i=1}^{M} a(i) \cdot c(i,k) = c(0,k), \quad k=1,\ldots,M
\]

where

\[
N-k-1 \quad c(i,k) = \sum_{j=0}^{N-k-1} s(j) \cdot s(j+k-i)
\]

are the covariance coefficients of \( s(*) \), \( N \) is the length of the input speech vector.

Note that the covariance matrix \( R \) is a symmetric matrix, and the equations can be solved efficiently with Cholesky decomposition method.


**Parameters** The function takes the following arguments:

- `coeff` The linear prediction coefficients.
- `signal` The input signal vector.
state Pointer to the internal state structure.

**Return Values** The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_SignalLPCCovarianceInit_F32(3MLIB), mlib_SignalLPCCovarianceFree_F32(3MLIB), attributes(5)
Name  mlib_SignalLPCCovarianceFree_S16, mlib_SignalLPCCovarianceFree_F32 – clean up for covariance method

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalLPCCovarianceFree_S16(void *state);
void mlib_SignalLPCCovarianceFree_F32(void *state);

Description  Each of these functions frees the internal state structure for covariance method of linear predictive coding (LPC).

This function cleans up the internal state structure and releases all memory buffers.

Parameters  Each of the functions takes the following arguments:

state  Pointer to the internal state structure.

Return Values  None.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPCCovarianceInit_S16(3MLIB),
mlib_SignalLPCCovarianceInit_F32(3MLIB),mlib_SignalLPCCovariance_S16(3MLIB),
mlib_SignalLPCCovariance_S16_Adq(3MLIB),mlib_SignalLPCCovariance_F32(3MLIB),
attributes(5)
Name  
mlib_SignalLPCCovarianceInit_S16, mlib_SignalLPCCovarianceInit_F32 – initialization for covariance method of linear predictive coding

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_SignalLPCCovarianceInit_S16(void *state,  
mlib_s32 length, mlib_s32 order);

mlib_status mlib_SignalLPCCovarianceInit_F32(void *state,  
mlib_s32 length, mlib_s32 order);

Description  
Each function initializes the internal state structure for covariance method of linear predictive coding (LPC).

The init function performs internal state structure allocation and global initialization. Per LPC function call initialization is done in LPC function, so the same internal state structure can be reused for multiple LPC function calls.

Parameters  
Each function takes the following arguments:

state  
Pointer to the internal state structure.

length  
The length of the input signal vector.

order  
The order of the linear prediction filter.

Return Values  
Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalLPCCovariance_S16(3MLIB), mlib_SignalLPCCovarianceFree_S16(3MLIB), attributes(5)
mlib_SignalLPCCovariance_S16, mlib_SignalLPCCovariance_S16_Adp – perform linear predictive coding with covariance method

Synopsis

c { flag... } file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLPCCovariance_S16(mlib_s16 *coeff,
        mlib_s32 cscale, const mlib_s16 *signal, void *state);

mlib_status mlib_SignalLPCCovariance_S16_Adp(mlib_s16 *coeff,
        mlib_s32 *cscale, const mlib_s16 *signal, void *state);

Description

Each function performs linear predictive coding with covariance method.

In linear predictive coding (LPC) model, each speech sample is represented as a linear combination of the past \( M \) samples.

\[
M \ 
s(n) = \sum_{i=1}^{M} a(i) \ast s(n-i) + G \ast u(n)
\]

where \( s(*) \) is the speech signal, \( u(*) \) is the excitation signal, and \( G \) is the gain constants, \( M \) is the order of the linear prediction filter. Given \( s(*) \), the goal is to find a set of coefficient \( a(*) \) that minimizes the prediction error \( e(*) \).

\[
M \ 
e(n) = s(n) - \sum_{i=1}^{M} a(i) \ast s(n-i)
\]

In covariance method, the coefficients can be obtained by solving following set of linear equations.

\[
M \ 
\sum_{i=1}^{M} a(i) \ast c(i,k) = c(0,k), k=1,\ldots,M
\]

where

\[
N-k-1 \ c(i,k) = \sum_{j=0}^{N-k-1} s(j) \ast s(j+k-i)
\]

are the covariance coefficients of \( s(*) \), \( N \) is the length of the input speech vector.

Note that the covariance matrix \( R \) is a symmetric matrix, and the equations can be solved efficiently with Cholesky decomposition method.

Note for functions with adaptive scaling (with \texttt{\_Adp} postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without \texttt{\_Adp} postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.

**Parameters**  Each function takes the following arguments:

- \texttt{coeff}  The linear prediction coefficients.
- \texttt{cscale}  The scaling factor of the linear prediction coefficients, where \texttt{actual\_data = output\_data \times 2^{\texttt{-scaling\_factor}})}.
- \texttt{signal}  The input signal vector with samples in Q15 format.
- \texttt{state}  Pointer to the internal state structure.

**Return Values**  Each function returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

**Attributes**  See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  \texttt{mlib\_SignalLPCCovariance\_Init\_S16(3MLIB)}, \texttt{mlib\_SignalLPCCovariance\_Free\_S16(3MLIB)}, \texttt{attributes(5)}
The `mlib_SignalLPCPerceptWeight_F32()` function performs perceptual weighting on input signal.

The perceptual weighting filter is defined as following.

\[
W(z) = \frac{A(z*r1)}{A(z*r2)}
\]

where \( A(z) \) is the inverse filter

\[
A(z) = 1 - \sum_{i=1}^{M} a(i) * z^{-i}
\]


**Parameters**
The function takes the following arguments:

- `sigwgt` The weighted signal vector.
- `signal` The input signal vector.
- `lpc` The linear prediction coefficients.
- `r1` The perceptual weighting filter coefficient, it is treated as 1 if 0 is supplied.
- `r2` The perceptual weighting filter coefficient, it is treated as 1 if 0 is supplied.
- `state` Pointer to the internal state structure.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_SignalLPCPerceptWeightInit_F32(3MLIB),
        mlib_SignalLPCPerceptWeightFree_F32(3MLIB), attributes(5)
include <mlib.h>

void mlib_SignalLPCPerceptWeightFree_S16(void *state);
void mlib_SignalLPCPerceptWeightFree_F32(void *state);

Each of these functions frees the internal state structure for perceptual weighting of linear predictive coding (LPC).

This function cleans up the internal state structure and releases all memory buffers.

Parameters Each of the functions takes the following arguments:

state Pointer to the internal state structure.

Return Values None.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalLPCPerceptWeightInit_S16(3MLIB),
mlib_SignalLPCPerceptWeightInit_F32(3MLIB),
mlib_SignalLPCPerceptWeight_S16(3MLIB),
mlib_SignalLPCPerceptWeight_F32(3MLIB), attributes(5)
**Name**  
mlib_SignalLPCPerceptWeightInit_S16, mlib_SignalLPCPerceptWeightInit_F32 – initialization for perceptual weighting of linear predictive coding

**Synopsis**  
c c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLPCPerceptWeightInit_S16(void *state,
        mlib_s32 length, mlib_s32 order);

mlib_status mlib_SignalLPCPerceptWeightInit_F32(void *state,
        mlib_s32 length, mlib_s32 order);

**Description**  
Each function initializes the internal state structure for perceptual weighting of linear predictive coding (LPC).

The init function performs internal state structure allocation and global initialization. Per LPC function call initialization is done in LPC function, so the same internal state structure can be reused for multiple LPC function calls.

**Parameters**  
Each function takes the following arguments:

- **state**  
  Pointer to the internal state structure.

- **length**  
  The length of the input signal vector.

- **order**  
  The order of the linear prediction filter.

**Return Values**  
Each function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalLPCPerceptWeight_S16(3MLIB),
mlib_SignalLPCPerceptWeightFree_S16(3MLIB), attributes(5)
mlib_SignalLPCPerceptWeight_S16 - perform perceptual weighting on input signal

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_SignalLPCPerceptWeight_S16(mlib_s16 *sigwgt, const mlib_s16 *signal, const mlib_s16 *lpc, mlib_s32 lscale, mlib_s16 r1, mlib_s16 r2, void *state);

Description

The mlib_SignalLPCPerceptWeight_S16() function performs perceptual weighting on input signal.

The perceptual weighting filter is defined as following.

\[ W(z) = \frac{A(z^*r1)}{A(z^*r2)} \]

where \( A(z) \) is the inverse filter

\[
A(z) = 1 - \sum_{i=1}^{M} a(i) * z^{-i}
\]


Parameters

The function takes the following arguments:

- **sigwgt**: The weighted signal vector, the signal samples are in Q15 format.
- **signal**: The input signal vector, the signal samples are in Q15 format.
- **lpc**: The linear prediction coefficients.
- **lscale**: The scaling factor of the linear prediction coefficients, where \( \text{actual\_data} = \text{input\_data} * 2^{\text{-scaling\_factor}} \).
- **r1**: The perceptual weighting filter coefficient, the coefficient is in Q15 format, it is treated as 1 if 0 is supplied.
- **r2**: The perceptual weighting filter coefficient, the coefficient is in Q15 format, it is treated as 1 if 0 is supplied.
- **state**: Pointer to the internal state structure.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
mlib_SignalLPCPerceptWeight_S16(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalLPCPerceptWeightInit_S16(3MLIB),  
mlib_SignalLPCPerceptWeightFree_S16(3MLIB), attributes(5)
The `mlib_SignalLPCPitchAnalyze_F32()` function performs open-loop pitch analysis.

The open-loop pitch analysis uses perceptual weighted signal and is done with following steps.

In the first step, three maxima of the correlation

\[
R(k) = \sum_{j=0}^{N-1} sw(j) \cdot sw(j-k)
\]

where \(N = \text{length}\), is located for each of the three search regions.

In the second step, the retained maxima \(R(T_i), i=0,1,2\) are normalized as following.

\[
Rn(t_i) = \frac{R(T_i)}{\sqrt{\sum_{j=0}^{N-1} (j-T_i)^2}} , \quad i=0,1,2
\]

where \(N = \text{length}\).

In the third step, the best open-loop delay \(T_{opt}\) is determined as following.

\[
T_{opt} = T_0 \quad \text{if} \quad (Rn(t_1) \geq 0.85 \cdot Rn(T_{opt}))
\]

\[
T_{opt} = t_1 \quad \text{if} \quad (Rn(t_2) \geq 0.85 \cdot Rn(T_{opt}))
\]

See G.729, G.729A, GSM EFR standards.

**Parameters**

The function takes the following arguments:

- `pitch` The speech pitch estimated.
- `sigwgt` The weighted signal vector. `sigwgt` points to the current sample of the weighted signal vector, `length` samples must be available after this point, and \(\text{MAX}(\text{region}[i], i=0,1,\ldots,5)\) samples must be available before this point.
- `region` The lower/upper boundaries of the three search regions, where `region[2*i]` is the lower boundary of search region \(i\) and `region[2*i+1]` is the upper boundary of search region \(i\).
length  The length of the signal vectors over which the correlation is calculated.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPCPitchAnalyze_S16(3MLIB), attributes(5)
The `mlib_SignalLPCPitchAnalyze_S16()` function performs open-loop pitch analysis. The open-loop pitch analysis uses perceptual weighted signal and is done with following steps.

In the first step, three maxima of the correlation

\[
R(k) = \sum_{j=0}^{N-1} sw(j) * sw(j-k)
\]

where \(N = \text{length}\), is located for each of the three search regions.

In the second step, the retained maxima \(R(T_i), i=0,1,2\) are normalized as following.

\[
Rn(ti) = \frac{R(T_i)}{\sqrt{\sum_{j=0}^{N-1} (sw(j-Ti))^2}}, i=0,1,2
\]

where \(N = \text{length}\).

In the third step, the best open-loop delay \(Topt\) is determined as following.

\[
Topt = T0 \\
\text{if } (Rn(t1) \geq (0.85 * Rn(Topt))) \text{ then } Topt = t1 \\
\text{if } (Rn(t2) \geq (0.85 * Rn(Topt))) \text{ then } Topt = t2
\]

See G.729, G.729A, GSM EFR standards.

**Parameters** The function takes the following arguments:

- `pitch` The speech pitch estimated.
- `sigwgt` The weighted signal vector with samples in Q15 format. `sigwgt` points to the current sample of the weighted signal vector, `length` samples must be available after this point, and \(\max\{\text{region}[i], i=0,1,\ldots,5\}\) samples must be available before this point.
- `region` The lower/upper boundaries of the three search regions, where `region[2*i]` is the lower boundary of search region \(i\) and `region[2*i+1]` is the upper boundary of search region \(i\).
length  The length of the signal vectors over which the correlation is calculated.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPCPitchAnalyze_F32(3MLIB), attributes(5)
The `mlib_SignalLSP2LPC_F32()` function converts line spectral pair coefficients to linear prediction coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

\[
P(z) = A(z) + z^{-1} A(z)
\]

\[
Q(z) = A(z) - z^{-1} A(z)
\]

where \( A(z) \) is the inverse filter

\[
A(z) = \sum_{i=1}^{M} a(i) z^{-i}
\]

Note that since \( P(z) \) is symmetric and \( Q(z) \) is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other. \( P(z) \) has a root at \( z = -1 \) (\( w = \pi \)) and \( Q(z) \) has a root at \( z = 1 \) (\( w = 0 \)).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

\[
q = \cos(w)
\]

where \( q \) is the LPS and \( w \) is the LPF.


### Parameters

- `lpc` The linear prediction coefficients.
- `lsp` The line spectral pair coefficients.
- `order` The order of the linear prediction filter.
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalLPC2LSP_F32(3MLIB), attributes(5)
mlib_SignalLSP2LPC_S16, mlib_SignalLSP2LPC_S16_Adp – convert line spectral pair coefficients to linear prediction coefficients

Synopsis
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalLSP2LPC_S16(mlib_s16 *lpc,
       mlib_s32 lscale, const mlib_s16 *lsp, mlib_s32 order);
mlib_status mlib_SignalLSP2LPC_S16_Adp(mlib_s16 *lpc,
       mlib_s32 *lscale, const mlib_s16 *lsp, mlib_s32 order);
```

Description
Each of the functions in this group converts line spectral pair coefficients to linear prediction coefficients.

The line spectral pair (LPS) coefficients are defined as the roots of the following two polynomials:

\[
P(z) = A(z) + \frac{1}{z} A(z^{-1})
\]

\[
Q(z) = A(z) - \frac{1}{z} A(z^{-1})
\]

where \( A(z) \) is the inverse filter

\[
A(z) = 1 - \sum_{i=1}^{M} a(i) \cdot \frac{1}{z}
\]

Note that since \( P(z) \) is symmetric and \( Q(z) \) is antisymmetric all roots of these polynomials are on the unit circle and they alternate each other. \( P(z) \) has a root at \( z = -1 \) \((w = \pi)\) and \( Q(z) \) has a root at \( z = 1 \) \((w = 0)\).

The line spectral frequency (LPF) are the angular frequency of the line spectral pair (LPS) coefficients.

\[
q = \cos(w)
\]

where \( q \) is the LPS and \( w \) is the LPF.


Note for functions with adaptive scaling (with _Adp postfix), the scaling factor of the output data will be calculated based on the actual data; for functions with non-adaptive scaling (without _Adp postfix), the user supplied scaling factor will be used and the output will be saturated if necessary.
Each function takes the following arguments:

- `lpc` The linear prediction coefficients.
- `lscale` The scaling factor of the line spectral pair coefficients, where \( \text{actual\_data} = \text{output\_data} \times 2^{\text{-scaling\_factor}} \).
- `lsp` The line spectral pair coefficients in Q15 format.
- `order` The order of the linear prediction filter.

Each function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_SignalLPC2LSP_S16(3MLIB), attributes(5)`
The `mlib_SignalMelCepstral_F32()` function performs cepstral analysis in mel frequency scale.

The first two steps of mel scale cepstral analysis is the same as in general cepstral analysis. After the logarithm of the spectrum magnitude is obtained, it is converted into mel frequency scale before the inverse Fourier transform.

\[
\begin{align*}
\text{Linear} & \quad \text{Inverse} \\
\text{...} & \quad \text{to} \\
X'(k) & \quad \text{Mel Scale} \\
& \quad X''(m) \\
& \quad \text{Transform} \\
& \quad c(n)
\end{align*}
\]

where \(X'(k)\) is defined in linear frequency scale and \(X''(m)\) is defined in mel frequency scale.

The mel frequency scale is defined as following.

\[
\text{freq}_\text{mel} = \text{melmul} \times \frac{\text{LOG10}(1 + \text{freq}_\text{linear} / \text{meldiv})}{\text{meldiv}}
\]

where \(\text{freq}_\text{mel}\) is the frequency in mel scale, \(\text{freq}_\text{linear}\) is the frequency in linear scale, \(\text{melmul}\) is the multiplying factor, \(\text{meldiv}\) is the dividing factor.

Optionally, a bank of band pass filters in linear frequency scale can be used below the bank of band pass filters in mel frequency scale, as shown below in linear frequency scale.

\[
0 \quad f_1 \quad f_2 \quad f_3 \quad f_p \quad f_{p+1} \quad f_{p+2} \quad f_{p+3} \quad f_{p+q} \\
\vdots \quad \vdots \quad \vdots \\
\text{...} \quad \text{...} \quad \text{...}
\]

where \(f_p = \text{melbgn}, f_{p+q} = \text{melend}, p = \text{nlinear}, q = \text{nmel} \); the filters number 1 to \(p\) are defined in linear frequency scale which have equal bandwidth in linear frequency scale; the filters number \(p+1\) to \(p+q\) are defined in mel frequency scale which have equal bandwidth in mel frequency scale and increasing bandwidth in linear frequency scale.


The function takes the following arguments:

- `cepst` The cepstral coefficients.
- `signal` The input signal vector.
- `state` Pointer to the internal state structure.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- mlib_SignalMelCepstralInit_F32(3MLIB), mlib_SignalMelCepstralFree_F32(3MLIB), attributes(5)
**Name**  
mlib_SignalMelCepstralFree_S16, mlib_SignalMelCepstralFree_F32 – clean up for cepstral analysis in mel frequency scale

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

void mlib_SignalMelCepstralFree_S16(void *state);  
void mlib_SignalMelCepstralFree_F32(void *state);

**Description**  
Each of these functions frees the internal `state` structure and releases all memory buffers for cepstral analysis in mel frequency scale.

**Parameters**  
Each of the functions takes the following arguments:

- `state` Pointer to the internal state structure.

**Return Values**  
None.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalMelCepstral_S16(3MLIB), mlib_SignalMelCepstral_F32(3MLIB), mlib_SignalMelCepstral_S16_Adp(3MLIB), mlib_SignalMelCepstralInit_S16(3MLIB), mlib_SignalMelCepstralInit_F32(3MLIB), attributes(5)
### Name
mlib_SignalMelCepstralInit_S16, mlib_SignalMelCepstralInit_F32 – initialization for cepstral analysis in mel frequency scale

### Synopsis
```
c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalMelCepstralInit_S16(void *state, mlib_s32 nlinear, mlib_s32 nmel, mlib_f32 melbgn, mlib_f32 melend, mlib_f32 meldiv, mlib_s32 order);
mlib_status mlib_SignalMelCepstralInit_F32(void *state, mlib_s32 nlinear, mlib_s32 nmel, mlib_f32 melbgn, mlib_f32 melend, mlib_f32 meldiv, mlib_s32 order);
```

### Description
Each of these functions initializes the internal state structure for cepstral analysis in mel frequency scale.

The init function performs internal state structure allocation and global initialization. Per function call initialization is done in each function, so the same internal state structure can be reused for multiple function calls.

### Parameters
Each of the functions takes the following arguments:
- **state**: Pointer to the internal state structure.
- **nlinear**: The number of band pass filters in linear frequency scale.
- **nmel**: The number of band pass filters in mel frequency scale.
- **melbgn**: The begin radian frequency of the mel scale filter bank defined in linear frequency scale, where $0 \leq \text{melbgn} < \text{melend} \leq \pi$. melbgn is ignored if nlinear = 0.
- **melend**: The end radian frequency of the mel scale filter bank defined in linear frequency scale, where $0 \leq \text{melbgn} < \text{melend} \leq \pi$.
- **meldiv**: The dividing factor in linear to mel scale conversion, linear scale is measured in radians, with PI corresponding to half the sampling rate.
- **order**: The order of the input signal vector and the cepstral coefficients, where \( \text{length} = 2^{\text{order}} \).

### Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also mlib_SignalMelCepstral_S16(3MLIB), mlib_SignalMelCepstral_F32(3MLIB),
mlib_SignalMelCepstral_S16_Adp(3MLIB), mlib_SignalMelCepstralFree_S16(3MLIB),
mlib_SignalMelCepstralFree_F32(3MLIB), attributes(5)
The `mlib_SignalMelCepstral_S16()` function performs cepstral analysis in mel frequency scale. The user supplied scaling factor will be used and the output will be saturated if necessary.

The first two steps of mel scale cepstral analysis is the same as in general cepstral analysis. After the logarithm of the spectrum magnitude is obtained, it is converted into mel frequency scale before the inverse Fourier transform.

\[
\begin{array}{c|c|c|c|c}
+-----------+ & +-----------+ \\
| Linear | | Inverse | \\
| \cdots | | \cdots | | Fourier | \\
X'(k) & Mel Scale & X''(m) & Transform & c(n) \\
\end{array}
\]

where \(X'(k)\) is defined in linear frequency scale and \(X''(m)\) is defined in mel frequency scale.

The mel frequency scale is defined as following.

\[
\text{freq}_\text{mel} = \text{melmul} \times \text{LOG10}(1 + \text{freq}_\text{linear} / \text{meldiv})
\]

where \(\text{freq}_\text{mel}\) is the frequency in mel scale, \(\text{freq}_\text{linear}\) is the frequency in linear scale, \(\text{melmul}\) is the multiplying factor, \(\text{meldiv}\) is the dividing factor.

Optionally, a bank of band pass filters in linear frequency scale can be used below the bank of band pass filters in mel frequency scale, as shown below in linear frequency scale.

\[
\begin{array}{cccccccc}
0 & f1 & f2 & f3 & fp & fp+1 & fp+2 & fp+3 & fp+q \\
\mid \cdots | \cdots | \cdots | \cdots | \cdots | \cdots | \cdots | \cdots | \cdots | \cdots |
\end{array} \rightarrow \text{freq}
\]

where \(fp = \text{melbgn}, \text{fp}+q = \text{melend}, \text{p} = \text{nlinear}, \text{q} = \text{nmel}\); the filters number 1 to \(p\) are defined in linear frequency scale which have equal bandwidth in linear frequency scale; the filters number \(p+1\) to \(p+q\) are defined in mel frequency scale which have equal bandwidth in mel frequency scale and increasing bandwidth in linear frequency scale.


The function takes the following arguments:

- `cepst` The cepstral coefficients.
- `cscale` The scaling factor of cepstral coefficients, where `actual_data = output_data * 2**(-scaling_factor)`.
- `signal` The input signal vector, the signal samples are in Q15 format.
- `state` Pointer to the internal state structure.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_SignalMelCepstralInit_S16(3MLIB)`, `mlib_SignalMelCepstral_S16_Adp(3MLIB)`, `mlib_SignalMelCepstralFree_S16(3MLIB)`, attributes(5)
The `mlib_SignalMelCepstral_S16_Adp()` function performs cepstral analysis in mel frequency scale. The scaling factor of the output data will be calculated based on the actual data.

The first two steps of mel scale cepstral analysis is the same as in general cepstral analysis. After the logarithm of the spectrum magnitude is obtained, it is converted into mel frequency scale before the inverse Fourier transform.

```
+-----------+ +-----------+
| Linear | | Inverse |
... ------->| to |------->| Fourier |------>
X'(k) | Mel Scale | X''(m) | Transform | c(n)
+-----------+ +-----------+
```

where $X'(k)$ is defined in linear frequency scale and $X''(m)$ is defined in mel frequency scale.

The mel frequency scale is defined as follows.

$$\text{freq}_\text{mel} = \text{melmul} \times \log_{10}(1 + \text{freq}_\text{linear} / \text{meldiv})$$

where $\text{freq}_\text{mel}$ is the frequency in mel scale, $\text{freq}_\text{linear}$ is the frequency in linear scale, $\text{melmul}$ is the multiplying factor, $\text{meldiv}$ is the dividing factor.

Optionally, a bank of band pass filters in linear frequency scale can be used below the bank of band pass filters in mel frequency scale, as shown below in linear frequency scale.

```
0 f1 f2 f3 fp fp+1 fp+2 fp+3 fp+q
|---|---|---| ... |---|-----|-----| ... | ... -> freq
```

where $fp = \text{melbgn}$, $fp+q = \text{melend}$, $p = \text{nlinear}$, $q = \text{nmel}$; the filters number 1 to $p$ are defined in linear frequency scale which have equal bandwidth in linear frequency scale; the filters number $p+1$ to $p+q$ are defined in mel frequency scale which have equal bandwidth in mel frequency scale and increasing bandwidth in linear frequency scale.


The function takes the following arguments:

- **cepst**: The cepstral coefficients.
- **cscale**: The scaling factor of cepstral coefficients, where \( \text{actual\_data} = \text{output\_data} \times 2^{-\text{scaling\_factor}} \).
- **signal**: The input signal vector, the signal samples are in Q15 format.
- **state**: Pointer to the internal state structure.

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_SignalMelCepstralInit_S16(3MLIB)`, `mlib_SignalMelCepstral_S16(3MLIB)`, `mlib_SignalMelCepstralFree_S16(3MLIB)`, attributes(5)
Name  mlib_SignalMerge_F32S_F32 – merge

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>

          mlib_status mlib_SignalMerge_F32S_F32(mlib_f32 *dst, const mlib_f32 *ch0,
                                         const mlib_f32 *ch1, mlib_s32 n);

Description  The mlib_SignalMerge_F32S_F32() function merges two signal arrays to form a stereo signal array.

Parameters  The function takes the following arguments:

  ch0  Input signal array of Channel 0.
  ch1  Input signal array of Channel 1.
  n    Number of samples in the source signal arrays.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMerge_S16S_S16(3MLIB), mlib_SignalSplit_F32_F32S(3MLIB),
           mlib_SignalSplit_S16_S16S(3MLIB), attributes(5)
The `mlib_SignalMerge_S16S_S16()` function merges two signal arrays to form a stereo signal array.

**Parameters**

- **dst**
- **ch0**
  - Input signal array of Channel 0.
- **ch1**
  - Input signal array of Channel 1.
- **n**
  - Number of samples in the source signal arrays.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_SignalMerge_F32S_F32(3MLIB)`, `mlib_SignalSplit_F32_F32S(3MLIB)`, `mlib_SignalSplit_S16_S16S(3MLIB)`, [attributes(5)]
Name

mlib_SignalMulBartlett_F32, mlib_SignalMulBartlett_F32S – Bartlett windowing multiplication

Synopsis

cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_SignalMulBartlett_F32(mlib_f32 *srcdst, mlib_s32 n);
mlib_status mlib_SignalMulBartlett_F32S(mlib_f32 *srcdst, mlib_s32 n);

Description

Each of these functions performs multiplication of the Bartlett window.

Parameters

Each of the functions takes the following arguments:

srcdst Input and output signal array.
n Number of samples in signal and window arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulBartlett_F32_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB), mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB), mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB), mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB), mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB), attributes(5)
mlib_SignalMulBartlett_F32_F32, mlib_SignalMulBartlett_F32S_F32S – Bartlett windowing multiplication

Synopsis
cc [ flag... ] file... -lmplib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulBartlett_F32_F32(mlib_f32 *dst,
const mlib_f32 *src, mlib_s32 n);

mlib_status mlib_SignalMulBartlett_F32S_F32S(mlib_f32 *dst,
const mlib_f32 *src, mlib_s32 n);

Description
Each of these functions performs multiplication of the Bartlett window.

Parameters
Each of the functions takes the following arguments:

- `dst` Output signal array.
- `src` Input signal array.
- `n` Number of samples in signal and window arrays.

Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB),
mlib_SignalMulBlackman_F32S_F32S(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB),
mlib_SignalMulHamming_F32S_F32S(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB),
mlib_SignalMulHanning_F32S_F32S(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
mlib_SignalMulKaiser_F32S_F32S(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB),
mlib_SignalMulRectangular_F32S_F32S(3MLIB), mlib_SignalMulWindow_F32(3MLIB),
mlib_SignalMulWindow_F32S_F32S(3MLIB), attributes(5)
mlib_SignalMulBartlett_S16, mlib_SignalMulBartlett_S16S – Bartlett windowing multiplication

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```
mlib_status mlib_SignalMulBartlett_S16(mlib_s16 *srcdst, mlib_s32 n);
mlib_status mlib_SignalMulBartlett_S16S(mlib_s16 *srcdst, mlib_s32 n);
```

Description  
Each of these functions performs multiplication of the Bartlett window.

Parameters  
Each of the functions takes the following arguments:
```
srcdst  Input and output signal array.
n  Number of samples in signal and window arrays.
```

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBlackman_S16_S16_S16(3MLIB),
mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulHamming_S16_S16_S16(3MLIB),
mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHanning_S16_S16_S16(3MLIB),
mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulKaiser_S16_S16_S16(3MLIB),
mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulRectangular_S16_S16_S16(3MLIB),
mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16_S16_S16(3MLIB),
mlib_SignalMulWindow_S16_S16(3MLIB), attributes(5)
Name  mlib_SignalMulBartlett_S16_S16, mlib_SignalMulBartlett_S16S_S16S – Bartlett windowing multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulBartlett_S16_S16(mlib_s16 *dst,
   const mlib_s16 *src, mlib_s32 n);
mlib_status mlib_SignalMulBartlett_S16S_S16S(mlib_s16 *dst,
   const mlib_s16 *src, mlib_s32 n);

Description  Each of these functions performs multiplication of the Bartlett window.

Parameters  Each of the functions takes the following arguments:

   dst    Output signal array.
   src    Input signal array.
   n      Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB),
mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulHamming_S16(3MLIB),
mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHanning_S16(3MLIB),
mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB),
mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB),
mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB),
attributes(5)
mlib_SignalMulBlackman_F32, mlib_SignalMulBlackman_F32S – Blackman windowing multiplication

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulBlackman_F32(mlib_f32 *srcdst, mlib_f32 alpha, mlib_s32 n);
mlib_status mlib_SignalMulBlackman_F32S(mlib_f32 *srcdst, mlib_f32 alpha, mlib_s32 n);

Description

Each of these functions performs multiplication of the Bartlett window.

Parameters

Each of the functions takes the following arguments:

- **srcdst** Input and output signal array.
- **alpha** Blackman window parameter. \(-1 < \alpha < 0\).
- **n** Number of samples in signal and window arrays.

Return Values

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB), mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB), mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB), mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB), attributes(5)
mlib_SignalMulBlackman_F32_F32

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulBlackman_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src, mlib_f32 alpha, mlib_s32 n);
mlib_status mlib_SignalMulBlackman_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, mlib_f32 alpha, mlib_s32 n);
```

Description

Each of these functions performs multiplication of the Bartlett window.

Parameters

Each of the functions takes the following arguments:

- `dst` Output signal array.
- `src` Input signal array.
- `alpha` Blackman window parameter. `-1 < alpha < 0`.
- `n` Number of samples in signal and window arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB),
mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB),
mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB),
mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB),
mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB),
mlib_SignalMulWindow_F32(3MLIB), attributes(5)
mlib_SignalMulBlackman_S16(3MLIB)

Name  mlib_SignalMulBlackman_S16, mlib_SignalMulBlackman_S16S – Blackman windowing multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>

          mlib_status mlib_SignalMulBlackman_S16(mlib_s16 *srcdst, mlib_f32 alpha,
                                          mlib_s32 n);
          mlib_status mlib_SignalMulBlackman_S16S(mlib_s16 *srcdst, mlib_f32 alpha,
                                           mlib_s32 n);

Description  Each of these functions performs multiplication of the Bartlett window.

Parameters  Each of the functions takes the following arguments:

  srcdst   Input and output signal array.
  alpha    Blackman window parameter. -1 < alpha < 0.
  n        Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB), mlib_SignalMulWindow_S16(3MLIB), attributes(5)
mlib_SignalMulBlackman_S16_S16, mlib_SignalMulBlackman_S16S_S16S – Blackman windowing multiplication

Synopsis

`cc [ flag... ] file... -lmlib [ library... ]`

```
#include <mlib.h>
mlib_status mlib_SignalMulBlackman_S16_S16(mlib_s16 *dst,
const mlib_s16 *src, mlib_f32 alpha, mlib_s32 n);
mlib_status mlib_SignalMulBlackman_S16S_S16S(mlib_s16 *dst,
const mlib_s16 *src, mlib_f32 alpha, mlib_s32 n);
```

Description Each of these functions performs multiplication of the Bartlett window.

Parameters Each of the functions takes the following arguments:

- **dst** Output signal array.
- **src** Input signal array.
- **alpha** Blackman window parameter. 
  \[-1 < \alpha < 0\].
- **n** Number of samples in signal and window arrays.

Return Values Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes See attributes(5) for descriptions of the following attributes:

```
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
```

See Also `mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHanning_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB), mlib_SignalMulWindow_S16(3MLIB), attributes(5)`
Name  mlib_SignalMul_F32, mlib_SignalMul_F32S – multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>

          mlib_status mlib_SignalMul_F32(mlib_f32 *src1dst,
             const mlib_f32 *src2, mlib_s32 n);
          mlib_status mlib_SignalMul_F32S(mlib_f32 *src1dst,
             const mlib_f32 *src2, mlib_s32 n);

Description  Each of these functions performs multiplication.

Parameters  Each of the functions takes the following arguments:
             
             src1dst  The first input and the output signal array.
             src2     The second input signal array.
             n        Number of samples in the input signal arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
Name  mlib_SignalMul_F32_F32, mlib_SignalMul_F32S_F32S – multiplication

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMul_F32_F32(mlib_f32 *dst,  
const mlib_f32 *src1, const mlib_f32 *src2, mlib_s32 n);
mlib_status mlib_SignalMul_F32S_F32S(mlib_f32 *dst,  
const mlib_f32 *src1, const mlib_f32 *src2, mlib_s32 n);

Description Each of these functions performs multiplication.

Parameters Each of the functions takes the following arguments:

- **src dst**  The output signal array.
- **src 1**  The first input signal array
- **src 2**  The second input signal array.
- **n**  Number of samples in the input signal arrays.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
mlib_SignalMulHamming_F32, mlib_SignalMulHamming_F32S – Bartlett windowing multiplication

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulHamming_F32(mlib_f32 *srcdst, mlib_s32 n);
mlib_status mlib_SignalMulHamming_F32S(mlib_f32 *srcdst, mlib_s32 n);

Description

Each of these functions performs multiplication of the Hamming window.

Parameters

Each of the functions takes the following arguments:

srcdst Input and output signal array.
n Number of samples in signal and window arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB),
mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB),
mlib_SignalMulHamming_F32_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB),
mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB),
mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB),
mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
Name | mlib_SignalMulHamming_F32_F32, mlib_SignalMulHamming_F32S_F32S – Hamming windowing multiplication

Synopsis | cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulHamming_F32_F32(mlib_f32 *dst,
const mlib_f32 *src, mlib_s32 n);
mlib_status mlib_SignalMulHamming_F32S_F32S(mlib_f32 *dst,
const mlib_f32 *src, mlib_s32 n);

Description | Each of these functions performs multiplication of the Hamming window.

Parameters | Each of the functions takes the following arguments:

dst | Output signal array.
src | Input signal array.
n | Number of samples in signal and window arrays.

Return Values | Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes | See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also | mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB), mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB), mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB), mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB), mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
Name  mlib_SignalMulHamming_S16, mlib_SignalMulHamming_S16S – Bartlett windowing multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulHamming_S16(mlib_s16 *srcdst, mlib_s32 n);
mlib_status mlib_SignalMulHamming_S16S(mlib_s16 *srcdst, mlib_s32 n);

Description  Each of these functions performs multiplication of the Hamming window.

Parameters  Each of the functions takes the following arguments:

srcdst  Input and output signal array.
n  Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB), mlib_SignalMulWindow_S16(3MLIB), attributes(5)
mlib_SignalMulHamming_S16_S16, mlib_SignalMulHamming_S16S_S16S – Hamming windowing multiplication

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulHamming_S16_S16(mlib_s16 *dst,  
   const mlib_s16 *src, mlib_s32 n);

mlib_status mlib_SignalMulHamming_S16S_S16S(mlib_s16 *dst,  
   const mlib_s16 *src, mlib_s32 n);
```

**Description**
Each of these functions performs multiplication of the Hamming window.

**Parameters**
Each of the functions takes the following arguments:
- `dst` Output signal array.
- `src` Input signal array.
- `n` Number of samples in signal and window arrays.

**Return Values**
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBartlett_S16_S16(3MLIB),  
mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB),  
mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB),  
mlib_SignalMulHanning_S16_S16S_S16S(3MLIB), mlib_SignalMulKaiser_S16(3MLIB),  
mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB),  
mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB),  
attributes(5)`
mlib_SignalMulHanning_F32(3MLIB)

Name mlib_SignalMulHanning_F32, mlib_SignalMulHanning_F32S – Hanning windowing multiplication

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulHanning_F32(mlib_f32 *srcdst, mlib_s32 n);
mlib_status mlib_SignalMulHanning_F32S(mlib_f32 *srcdst, mlib_s32 n);

Description Each of these functions performs multiplication of the Hanning window.

Parameters Each of the functions takes the following arguments:

srcdst Source and destination signal array.
n Number of samples in signal and window arrays.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB), mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB), mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB), mlib_SignalMulHamming_F32_F32_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB), mlib_SignalMulKaiser_F32_F32_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulRectangular_F32_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
mlib_SignalMulHanning_F32_F32

Name  
mlib_SignalMulHanning_F32_F32, mlib_SignalMulHanning_F32S_F32S
Hanning windowing multiplication

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_SignalMulHanning_F32_F32(mlib_f32 *dst,
          mlib_s32 const mlib_f32 *src, n);
mlib_status mlib_SignalMulHanning_F32S_F32S(mlib_f32 *dst,
          mlib_s32 const mlib_f32 *src, mlib_s32 n);

Description  
Each of these functions performs multiplication of the Hanning window.

Parameters  
Each of the functions takes the following arguments:

dst  Destination signal array.
src  Source signal array.
n  Number of samples in signal and window arrays.

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB),
mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB),
mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB),
mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB),
mlib_SignalMulWindow_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
Name  mlib_SignalMulHanning_S16, mlib_SignalMulHanning_S16S – Hanning windowing multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulHanning_S16(mlib_s16 *srcdst, mlib_s32 n);
mlib_status mlib_SignalMulHanning_S16S(mlib_s16 *srcdst, mlib_s32 n);

Description  Each of these functions performs multiplication of the Hanning window.

Parameters  Each of the functions takes the following arguments:

srcdst  Source and destination signal array.

n  Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB),
mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB),
mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHamming_S16(3MLIB),
mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB),
mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB),
mlib_SignalMulRectangular_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB),
mlib_SignalMulWindow_S16(3MLIB), attributes(5)
mlib_SignalMulHanning_S16_S16(3MLIB)

**Name**  
mlib_SignalMulHanning_S16_S16, mlib_SignalMulHanning_S16S_S16S – Hanning windowing multiplication

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_SignalMulHanning_S16_S16(mlib_s16 *dst,
const mlib_s16 *src, mlib_s32 n);
mlib_status mlib_SignalMulHanning_S16S_S16S(mlib_s16 *dst,
const mlib_s16 *src, mlib_s32 n);
```

**Description**  
Each of these functions performs multiplication of the Hanning window.

**Parameters**  
Each of the functions takes the following arguments:

- `dst` Destination signal array.
- `src` Source signal array.
- `n` Number of samples in signal and window arrays.

**Return Values**  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB), attributes(5)
mlib_SignalMulKaiser_F32, mlib_SignalMulKaiser_F32S – Kaiser windowing multiplication

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulKaiser_F32(mlib_f32 *srdst, mlib_f32 beta,
                                   mlib_s32 n);
mlib_status mlib_SignalMulKaiser_F32S(mlib_f32 *srdst, mlib_f32 beta,
                                   mlib_s32 n);

Description

Each of these functions performs multiplication of the Kaiser window.

Parameters

Each of the functions takes the following arguments:

- srdst Source and destination signal array.
- beta Kaiser window parameter.
- n Number of samples in signal and window arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB), mlib_SignalMulBlackman_F32(3MLIB),
mlib_SignalMulBlackman_F32_F32(3MLIB), mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB),
mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB),
attributes(5)
### Name
mlib_SignalMulKaiser_F32_F32, mlib_SignalMulKaiser_F32S_F32S – Kaiser windowing multiplication

### Synopsis
```c
#include <mlib.h>

mlib_status mlib_SignalMulKaiser_F32_F32(mlib_f32 *dst, 
              const mlib_f32 *src, mlib_f32 beta, mlib_s32 n);

mlib_status mlib_SignalMulKaiser_F32S_F32S(mlib_f32 *dst, 
              const mlib_f32 *src, mlib_f32 beta, mlib_s32 n);
```

### Description
Each of these functions performs multiplication of the Kaiser window.

### Parameters
Each of the functions takes the following arguments:
- `dst` Output signal array.
- `src` Input signal array.
- `beta` Kaiser window parameter.
- `n` Number of samples in signal and window arrays.

### Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB), mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB), mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB), mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB), mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB), mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
Each of these functions performs multiplication of the Kaiser window.

**Parameters**

Each of the functions takes the following arguments:

- `srcdst` Source and destination signal array.
- `beta` Kaiser window parameter.
- `n` Number of samples in signal and window arrays.

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_SignalMulBartlett_S16(3MLIB)`
- `mlib_SignalMulBartlett_S16_S16(3MLIB)`
- `mlib_SignalMulBlackman_S16(3MLIB)`
- `mlib_SignalMulBlackman_S16_S16(3MLIB)`
- `mlib_SignalMulHamming_S16(3MLIB)`
- `mlib_SignalMulHamming_S16_S16(3MLIB)`
- `mlib_SignalMulHanning_S16(3MLIB)`
- `mlib_SignalMulHanning_S16_S16(3MLIB)`
- `mlib_SignalMulKaiser_S16_S16(3MLIB)`
- `mlib_SignalMulRectangular_S16(3MLIB)`
- `mlib_SignalMulRectangular_S16_S16(3MLIB)`
- `mlib_SignalMulWindow_S16_S16(3MLIB)`
- attributes(5)
mlib_SignalMulKaiser_S16_S16(3MLIB)

Name mlib_SignalMulKaiser_S16_S16, mlib_SignalMulKaiser_S16S_S16S – Kaiser windowing multiplication

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulKaiser_S16_S16(mlib_s16 *dst,
const mlib_s16 *src, mlib_f32 beta, mlib_s32 n);
mlib_status mlib_SignalMulKaiser_S16S_S16S(mlib_s16 *dst,
const mlib_s16 *src, mlib_f32 beta, mlib_s32 n);

Description Each of these functions performs multiplication of the Kaiser window.

Parameters Each of the functions takes the following arguments:

dst Output signal array.
src Input signal array.
beta Kaiser window parameter.
n Number of samples in signal and window arrays.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBartlett_S16_S16(3MLIB),
mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB),
mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB),
mlib_SignalMulHanning_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB),
mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16(3MLIB),
mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16(3MLIB),
mlib_SignalMulWindow_S16_S16(3MLIB), attributes(5)
mlib_SignalMulRectangular_F32, mlib_SignalMulRectangular_F32S – rectangular windowing multiplication

Synopsis

```c
#include <mlib.h>

mlib_status mlib_SignalMulRectangular_F32(mlib_f32 *srcdst, mlib_s32 m, mlib_s32 n);
mlib_status mlib_SignalMulRectangular_F32S(mlib_f32 *srcdst, mlib_s32 m, mlib_s32 n);
```

Description

Each of these functions performs multiplication of the rectangular window.

Parameters

Each of the functions takes the following arguments:

- `srcdst` Input and output signal array.
- `m` Rectangular window parameter.
- `n` Number of samples in signal and window arrays.

Return Values

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB),
- mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB),
- mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB),
- mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB),
- mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
- mlib_SignalMulRectangular_F32_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB),
- mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
Name  mlib_SignalMulRectangular_F32_F32, mlib_SignalMulRectangular_F32S_F32S – rectangular windowing multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulRectangular_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src, mlib_s32 m, mlib_s32 n);

mlib_status mlib_SignalMulRectangular_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, mlib_s32 m, mlib_s32 n);

Description  Each of these functions performs multiplication of the Hamming window.

Parameters  Each of the functions takes the following arguments:

dst  Output signal array.

src  Input signal array.

m  Rectangular window parameter.

n  Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_F32(3MLIB), mlib_SignalMulBartlett_F32_F32(3MLIB),
mlib_SignalMulBlackman_F32(3MLIB), mlib_SignalMulBlackman_F32_F32(3MLIB),
mlib_SignalMulHamming_F32(3MLIB), mlib_SignalMulHamming_F32_F32(3MLIB),
mlib_SignalMulHanning_F32(3MLIB), mlib_SignalMulHanning_F32_F32(3MLIB),
mlib_SignalMulKaiser_F32(3MLIB), mlib_SignalMulKaiser_F32_F32(3MLIB),
mlib_SignalMulRectangular_F32(3MLIB), mlib_SignalMulWindow_F32(3MLIB),
mlib_SignalMulWindow_F32_F32(3MLIB), attributes(5)
mlib_SignalMulRectangular_S16(3MLIB)

Name  mlib_SignalMulRectangular_S16, mlib_SignalMulRectangular_S16S – rectangular windowing multiplication

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulRectangular_S16(mlib_s16 *srcdst, mlib_s32 m, mlib_s32 n);
mlib_status mlib_SignalMulRectangular_S16S(mlib_s16 *srcdst, mlib_s32 m, mlib_s32 n);

Description  Each of these functions performs multiplication of the rectangular window.

Parameters  Each of the functions takes the following arguments:
srcdst  Input and output signal array.
m  Rectangular window parameter.
n  Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB),
mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB),
mlib_SignalMulHamming_S16_S16(3MLIB), mlib_SignalMulHamming_S16(3MLIB),
mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulHanning_S16(3MLIB),
mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB),
mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16(3MLIB),
mlib_SignalMulWindow_S16_S16(3MLIB), attributes(5)
mlib_SignalMulRectangular_S16_S16

### Name
mlib_SignalMulRectangular_S16_S16, mlib_SignalMulRectangular_S16S_S16S – rectangular windowing multiplication

### Synopsis
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulRectangular_S16_S16(mlib_s16 *dst,
    const mlib_s16 *src, mlib_s32 m, mlib_s32 n);
mlib_status mlib_SignalMulRectangular_S16S_S16S(mlib_s16 *dst,
    const mlib_s16 *src, mlib_s32 m, mlib_s32 n);
```

### Description
Each of these functions performs multiplication of the Hamming window.

### Parameters
Each of the functions takes the following arguments:
- `dst` Output signal array.
- `src` Input signal array.
- `m` Rectangular window parameter.
- `n` Number of samples in signal and window arrays.

### Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_SignalMulBartlett_S16_S16(3MLIB), mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB), mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB), mlib_SignalMulHanning_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB), mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB), mlib_SignalMulWindow_S16(3MLIB), attributes(5)`. 

Multimedia Library Functions - Part 5

1079
mlib_SignalMul_S16_S16_Sat, mlib_SignalMul_S16S_S16S_Sat – multiplication

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMul_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src1, const mlib_s16 *src2, mlib_s32 n);
mlib_status mlib_SignalMul_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src1, const mlib_s16 *src2, mlib_s32 n);

Description

Each of these functions performs multiplication.

Parameters

Each of the functions takes the following arguments:

dst  Output signal array.
src1 The first input signal array.
src2 The second input signal array.
n    Number of samples in the input signal arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMul_S16_Sat(3MLIB), attributes(5)
mlib_SignalMul_S16_Sat, mlib_SignalMul_S16S_Sat – multiplication

Synopsis

```
cc [ flag... ] file... -lmllib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMul_S16_Sat(mlib_s16 *srcldst,
        const mlib_s16 *src2, mlib_s32 n);
mlib_status mlib_SignalMul_S16S_Sat(mlib_s16 *srcldst,
        const mlib_s16 *src2, mlib_s32 n);
```

Description

Each of these functions performs multiplication.

Parameters

Each of the functions takes the following arguments:

- `srcldst` The first input and the output signal array.
- `src2` The second input signal array.
- `n` Number of samples in the input signal arrays.

Return Values

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMul_S16_S16_Sat(3MLIB), attributes(5)
Name  mlib_SignalMulSAdd_F32, mlib_SignalMulSAdd_F32S – multiplication by a scalar plus addition

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulSAdd_F32(mlib_f32 *scr1dst,
const mlib_f32 *const mlib_f32 *src2, c, mlib_s32 n);

mlib_status mlib_SignalMulSAdd_F32S(mlib_f32 *scr1dst,
const mlib_f32 *const mlib_f32 *src2, const mlib_f32 *c,
mlib_s32 n);

Description  Each of these functions performs multiplication by a scalar plus addition.

Parameters  Each of the functions takes the following arguments:

- src1dst  The first input and the output signal array.
- src2  The second input signal array.
- c  Scaling factor. In the stereo version, c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- n  Number of samples in the input signal arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulSAdd_F32_F32(3MLIB), attributes(5)
**Name**
mlib_SignalMulSAdd_F32_F32, mlib_SignalMulSAdd_F32S_F32S – multiplication by a scalar plus addition

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulSAdd_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src1, const mlib_f32 *src2, const mlib_f32 *c,
    mlib_s32 n);

mlib_status mlib_SignalMulSAdd_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src1, const mlib_f32 *src2, const mlib_f32 *c,
    mlib_s32 n);

**Description**
Each of these functions performs multiplication by a scalar.

**Parameters**
Each of the functions takes the following arguments:
- `dst` Output signal array.
- `src1` The first input signal array.
- `src2` The second input signal array.
- `c` Scaling factor. In the stereo version, `c[0]` contains the scaling factor for channel 0, and `c[1]` holds the scaling factor for channel 1.
- `n` Number of samples in the input signal arrays.

**Return Values**
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalMulSAdd_F32_F32(3MLIB), attributes(5)
mlib_SignalMulSAdd_S16_S16_Sat, mlib_SignalMulSAdd_S16S_S16S_Sat – multiplication by a scalar plus addition

**Synopsis**

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulSAdd_S16_S16_Sat(mlib_s16 *dst,  
    const mlib_s16 *src1, const mlib_s16 *src2, const mlib_s16 *c,  
    mlib_s32 n);

mlib_status mlib_SignalMulSAdd_S16S_S16S_Sat(mlib_s16 *dst,  
    const mlib_s16 *src1, const mlib_s16 *src2, const mlib_s16 *c,  
    mlib_s32 n);

**Description**

Each of these functions performs multiplication by a scalar.

**Parameters**

Each of the functions takes the following arguments:

- **dst**: Output signal array.
- **src1**: The first input signal array.
- **src2**: The second input signal array.
- **c**: Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- **n**: Number of samples in the input signal arrays.

**Return Values**

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_SignalMulSAdd_S16_S16_Sat(3MLIB), attributes(5)
mlib_SignalMulSAdd_S16_Sat, mlib_SignalMulSAdd_S16S_Sat – multiplication by a scalar plus addition

**Synopsis**
```c
#include <mlib.h>

mlib_status mlib_SignalMulSAdd_S16_Sat(mlib_s16 *src1dst,
    const mlib_s16 *src2, const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_SignalMulSAdd_S16S_Sat(mlib_s16 *src1dst,
    const mlib_s16 *src2, const mlib_s16 *c, mlib_s32 n);
```

**Description**
Each of these functions performs multiplication by a scalar plus addition.

**Parameters**
Each of the functions takes the following arguments:
- `src1dst` The first input and the output signal array.
- `src2` The second input signal array.
- `c` Scaling factor. The scaling factor is in Q15 format. In the stereo version; `c[0]` contains the scaling factor for channel 0, and `c[1]` holds the scaling factor for channel 1.
- `n` Number of samples in the input signal arrays.

**Return Values**
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalMulSAdd_S16_S16_Sat(3MLIB), attributes(5)
Name  
mlib_SignalMulS_F32, mlib_SignalMulS_F32S – multiplication by a scalar

Synopsis 
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalMulS_F32(mlib_f32 *srcdst, const mlib_f32 *c,
mlib_s32 n);
mlib_status mlib_SignalMulS_F32S(mlib_f32 *srcdst, const mlib_f32 *c,
mlib_s32 n);

Description  
Each of these functions performs multiplication by a scalar.

Parameters  
Each of the functions takes the following arguments:

srcdst  
Input and output signal array.

c  
Scaling factor. In the stereo version, c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.

n  
Number of samples in the input signal arrays.

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalMulS_F32_F32(3MLIB), attributes(5)
mlib_SignalMulS_F32_F32, mlib_SignalMulS_F32S_F32S – multiplication by a scalar

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulS_F32_F32(mlib_f32 *dst, const mlib_f32 *src, const mlib_f32 *c, mlib_s32 n);
mlib_status mlib_SignalMulS_F32S_F32S(mlib_f32 *dst, const mlib_f32 *src, const mlib_f32 *c, mlib_s32 n);

Description
Each of these functions performs multiplication by a scalar.

Parameters
Each of the functions takes the following arguments:

- dst: Output signal array.
- src: Input signal array.
- c: Scaling factor. In the stereo version, c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- n: Number of samples in the input signal arrays.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalMulS_F32_F32(3MLIB), attributes(5)
Name  mlib_SignalMulShift_S16_S16_Sat, mlib_SignalMulShift_S16S_S16S_Sat – multiplication with shifting

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulShift_S16_S16_Sat(mlib_s16 *dst,
   const mlib_s16 *src1, const mlib_s16 *src2,
   mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalMulShift_S16S_S16S_Sat(mlib_s16 *dst,
   const mlib_s16 *src1, const mlib_s16 *src2,
   mlib_s32 shift, mlib_s32 n);

Description  Each of these functions performs multiplication with shifting.

Parameters  Each of the functions takes the following arguments:

dst  Output signal array.
src1  The first input signal array.
src2  The second input signal array.
shift  Left shifting factor.
n  Number of samples in the input signal arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulShift_S16_Sat(3MLIB), attributes(5)
mlib_SignalMulShift_S16_Sat, mlib_SignalMulShift_S16S_Sat – multiplication with shifting

Synopsis

cc [ flag ... ] file ... -lmlib [ library ... ]

#include <mlib.h>

mlib_status mlib_SignalMulShift_S16_Sat(mlib_s16 *src1dst, const mlib_s16 *src2, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalMulShift_S16S_Sat(mlib_s16 *src1dst, const mlib_s16 *src2, mlib_s32 shift, mlib_s32 n);

Description

Each of these functions performs multiplication with shifting.

Parameters

Each of the functions takes the following arguments:

- **src1dst** The first input and the output signal array.
- **src2** The second input signal array.
- **shift** Left shifting factor.
- **n** Number of samples in the input signal arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulShift_S16_Sat(3MLIB), attributes(5)
Name: mlib_SignalMulS_S16_S16_Sat, mlib_SignalMulS_S16S_S16S_Sat – multiplication by a scalar

Synopsis: 
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulS_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_SignalMulS_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *c, mlib_s32 n);
```

Description: Each of these functions performs multiplication by a scalar.

Parameters: Each of the functions takes the following arguments:
- `dst` Output signal array.
- `src` Input signal array.
- `c` Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- `n` Number of samples in the input signal arrays.

Return Values: Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_SignalMulS_S16_Sat(3MLIB), attributes(5)
**Name**
mlib_SignalMulS_S16_Sat, mlib_SignalMulS_S16S_Sat – multiplication by a scalar

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_SignalMulS_S16_Sat(mlib_s16 *srcdst, const mlib_s16 *c,
                                   mlib_s32 n);
mlib_status mlib_SignalMulS_S16S_Sat(mlib_s16 *srcdst, const mlib_s16 *c,
                                   mlib_s32 n);
```

**Description**
Each of these functions performs multiplication by a scalar.

**Parameters**
Each of the functions takes the following arguments:
- `srcdst` Input and output signal array.
- `c` Scaling factor. The scaling factor is in Q15 format. In the stereo version; `c[0]` contains the scaling factor for channel 0, and `c[1]` holds the scaling factor for channel 1.
- `n` Number of samples in the input signal arrays.

**Return Values**
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalMulS_S16_S16_Sat(3MLIB), attributes(5)
mlib_SignalMulSShiftAdd_S16_S16_Sat

**Name**
mlib_SignalMulSShiftAdd_S16_S16_Sat, mlib_SignalMulSShiftAdd_S16S_S16S_Sat – multiplication by a scalar plus addition

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulSShiftAdd_S16_S16_Sat(mlib_s16 *dst,
const mlib_s16 *src1, const mlib_s16 *src2, const mlib_s16 *c,
mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalMulSShiftAdd_S16S_S16S_Sat(mlib_s16 *dst,
const mlib_s16 *src1, const mlib_s16 *src2, const mlib_s16 *c,
mlib_s32 shift, mlib_s32 n);

**Description**
Each of these functions performs multiplication by a scalar with shifting plus addition.

**Parameters**
Each of the functions takes the following arguments:

- **dst**
  Output signal array.

- **src1**
  The first input signal array.

- **src2**
  The second input signal array.

- **c**
  Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.

- **shift**
  Left shifting factor.

- **n**
  Number of samples in the input signal arrays.

**Return Values**
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalMulSShiftAdd_S16_S16_Sat(3MLIB), attributes(5)
mlib_SignalMulSShiftAdd_S16_Sat, mlib_SignalMulSShiftAdd_S16S_Sat – multiplication by a scalar plus addition

Synopsis

```c
#include <mlib.h>

mlib_status mlib_SignalMulSShiftAdd_S16_Sat(mlib_s16 *scr1dst, const mlib_s16 *src2, const mlib_s16 *c, mlib_s32 shift, mlib_s32 n);

mlib_status mlib_SignalMulSShiftAdd_S16S_Sat(mlib_s16 *scr1dst, const mlib_s16 *src2, const mlib_s16 *c, mlib_s32 shift, mlib_s32 n);
```

Description

Each of these functions performs multiplication by a scalar with shifting plus addition.

Parameters

Each of the functions takes the following arguments:

- `src1dst` The first input and the output signal array.
- `src2` The second input signal array.
- `c` Scaling factor. The scaling factor is in Q15 format. In the stereo version; `c[0]` contains the scaling factor for channel 0, and `c[1]` holds the scaling factor for channel 1.
- `n` Number of samples in the input signal arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulSShiftAdd_S16_S16_Sat(3MLIB), attributes(5)
Name  mlib_SignalMulSShift_S16_S16_Sat, mlib_SignalMulSShift_S16S_S16S_Sat – multiplication by a scalar with shifting.

Synopsis  cc [ flag... ] file... -lmllib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulSShift_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *c, mlib_s32 shift,
    mlib_s32 n);

mlib_status mlib_SignalMulSShift_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *c, mlib_s32 shift,
    mlib_s32 n);

Description  Each of these functions performs multiplication by a scalar with shifting.

Parameters  Each of the functions takes the following arguments:

dst  Destination signal array.
src  Source signal array.
c  Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains
the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
shift  Left shifting factor.
n  Number of samples in the input signal arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulSShift_S16_S16_Sat(3MLIB), attributes(5)
mlib_SignalMulSShift_S16_Sat, mlib_SignalMulSShift_S16S_Sat – multiplication by a scalar with shifting

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalMulSShift_S16_Sat(mlib_s16 *srcdst,
    const mlib_s16 *c, mlib_s32 shift, mlib_s32 n);
mlib_status mlib_SignalMulSShift_S16S_Sat(mlib_s16 *srcdst,
    const mlib_s16 *c, mlib_s32 shift, mlib_s32 n);
```

Description

Each of these functions performs multiplication by a scalar with shifting.

Parameters

Each of the functions takes the following arguments:

- `srcdst` Source and destination signal array.
- `c` Scaling factor. The scaling factor is in Q15 format. In the stereo version; c[0] contains the scaling factor for channel 0, and c[1] holds the scaling factor for channel 1.
- `shift` Left shifting factor.
- `n` Number of samples in the input signal arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulSShift_S16_S16_Sat(3MLIB), attributes(5)
mlib_SignalMulWindow_F32 – windowing

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulWindow_F32(mlib_f32 *srcdst,
   const mlib_f32 *window, mlib_s32 n);

Description
The mlib_SignalMulWindow_F32() function performs a windowing operation.

Parameters
The function takes the following arguments:

srcdst Input and output signal array.

window Window coefficient array. The window coefficients are in Q15 format.

n Number of samples in signal and window arrays.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
attributes(5)
The `mlib_SignalMulWindow_F32_F32()` function performs a windowing operation.

### Parameters

- **dst**
  - Output signal array.
- **src**
  - Input signal array.
- **window**
  - Window coefficient array. The window coefficients are in Q15 format.
- **n**
  - Number of samples in signal and window arrays.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `attributes(5)`
The `mlib_SignalMulWindow_F32S()` function performs a windowing operation.

**Parameters**
- `srcdst` Input and output signal array are in stereo format where `srcdst[0]` contains the values for channel 0, and `srcdst[1]` holds the values for channel 1.
- `window` Window coefficient array. The window coefficients are in Q15 format.
- `n` Number of samples in signal and window arrays.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
attributes(5)
mlib_SignalMulWindow_F32S_F32S() function performs a windowing operation.

Parameters

Name | Description
--- | ---
dst | Output signal array is in stereo format where dst[0] contains the values for channel 0, and dst[1] holds the values for channel 1.
src | Input signal array is in stereo format where src[0] contains the values for channel 0, and src[1] holds the values for channel 1.
window | Window coefficient array. The window coefficients are in Q15 format.
n | Number of samples in signal and window arrays.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

attributes(5)
mlib_SignalMulWindow_S16(3MLIB)

Name  mlib_SignalMulWindow_S16, mlib_SignalMulWindow_S16S – windowing

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulWindow_S16(mlib_s16 *srcdst,
   const mlib_s16 *window, mlib_s32 n);
mlib_status mlib_SignalMulWindow_S16S(mlib_s16 *srcdst,
   const mlib_s16 *window, mlib_s32 n);

Description  Each of these functions performs a windowing operation.

Parameters  Each of the functions takes the following arguments:

srcdst    Input and output signal array.

window    Window coefficient array. The window coefficients are in Q15 format.

n         Number of samples in signal and window arrays.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBartlett_S16_S16(3MLIB),
mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB),
mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB),
mlib_SignalMulHanning_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB),
mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB),
mlib_SignalMulRectangular_S16(3MLIB),
mlib_SignalMulRectangular_S16_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB),
attributes(5)
mlib_SignalMulWindow_S16_S16, mlib_SignalMulWindow_S16S_S16S – windowing

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalMulWindow_S16_S16(mlib_s16 *dst,
        const mlib_s16 *src, const mlib_s16 *window, mlib_s32 n);
mlib_status mlib_SignalMulWindow_S16S_S16S(mlib_s16 *dst,
        const mlib_s16 *src, const mlib_s16 *window, mlib_s32 n);

Description

Each of these functions performs a windowing operation.

Parameters

Each of the functions takes the following arguments:

 dst Output signal array.
 src Input signal array.
 window Window coefficient array. The window coefficients are in Q15 format.
 n Number of samples in signal and window arrays.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_SignalMulBartlett_S16(3MLIB), mlib_SignalMulBartlett_S16_S16(3MLIB),
mlib_SignalMulBlackman_S16(3MLIB), mlib_SignalMulBlackman_S16_S16(3MLIB),
mlib_SignalMulHamming_S16(3MLIB), mlib_SignalMulHamming_S16_S16(3MLIB),
mlib_SignalMulHanning_S16(3MLIB), mlib_SignalMulHanning_S16_S16(3MLIB),
mlib_SignalMulKaiser_S16(3MLIB), mlib_SignalMulKaiser_S16_S16(3MLIB),
mlib_SignalMulRectangular_S16(3MLIB), mlib_SignalMulRectangular_S16_S16(3MLIB),
mlib_SignalMulWindow_S16(3MLIB), mlib_SignalMulWindow_S16_S16(3MLIB), attributes(5)
mlib_SignalNLMSFilter, mlib_SignalNLMSFilterInit_S16_S16,
mlib_SignalNLMSFilterInit_S16S_S16S, mlib_SignalNLMSFilterInit_F32_F32,
mlib_SignalNLMSFilterInit_F32S_F32S, mlib_SignalNLMSFilter_S16_S16_Sat,
mlib_SignalNLMSFilter_S16S_S16S_Sat, mlib_SignalNLMSFilter_F32_F32,
mlib_SignalNLMSFilter_F32S_F32S, mlib_SignalNLMSFilterNonAdapt_S16_S16_Sat,
mlib_SignalNLMSFilterNonAdapt_S16S_S16S_Sat,
mlib_SignalNLMSFilterNonAdapt_F32_F32, mlib_SignalNLMSFilterNonAdapt_F32S_F32S,
mlib_SignalNLMSFilterFree_S16_S16, mlib_SignalNLMSFilterFree_S16S_S16S,
mlib_SignalNLMSFilterFree_F32_F32, mlib_SignalNLMSFilterFree_F32S_F32S –
normalized least mean square (NLMS) adaptive filtering

Synopsis

cc [-flag...] file... -lmlib [ library... ]

#include <mlib.h>

mlib_status mlib_SignalNLMSFilterInit_S16_S16(void **filter,
    const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilterInit_S16S_S16S(void **filter,
    const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilterInit_F32_F32(void **filter,
    const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilterInit_F32S_F32S(void **filter,
    const mlib_f32 *flt, mlib_s32 tap, mlib_f32 beta);

mlib_status mlib_SignalNLMSFilter_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilter_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilter_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilter_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, const mlib_s16 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

mlib_status mlib_SignalNLMSFilterNonAdapt_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, const mlib_f32 *ref, void *filter, mlib_s32 n);

void mlib_SignalNLMSFilterFree_S16_S16(void *filter);

void mlib_SignalNLMSFilterFree_S16S_S16S(void *filter);

void mlib_SignalNLMSFilterFree_F32_F32(void *filter);
void mlib_SignalNLMSFilterFree_F32S_F32S(void *filter);

**Description**
The normalized LMS adaptive algorithm is summarized as follows:

1. Initialize the weights \( W_k(i), i = 0, 1, \ldots, \text{tap} - 1 \).
2. Initialize previous source elements \( X_0(i), i = 0, 1, \ldots, \text{tap} - 1 \).
3. Read \( X_k(t) \) from \( \text{src} \) and \( Y_k(t) \) from \( \text{ref} \), \( t = 0, 1, \ldots, n - 1 \).
4. Compute filter output: \( n_k = \sum(W_k(i) \times X_k(t - i)), i = 0, 1, \ldots, \text{tap} - 1 \). If \( i > t \), use previous source elements stored in the \( X_0 \) vector.
5. Compute source elements power: \( P_{wk} = \sum(X_k(t - i) \times X_k(t - i)), i = 0, 1, \ldots, \text{tap} - 1 \). If \( i > t \), use previous source elements stored in the \( X_0 \) vector.
6. Store filter output: \( \text{dst}[t] = n_k \).
7. Compute the error estimate: \( E_k = Y_k - n_k \).
8. Compute factor \( BE_0 = 2 \times \beta \times E_k / P_{wk} \).
9. Update filter weights: \( W_k(i) += BE_0 \times X_k(t - i), i = 0, 1, \ldots, \text{tap} - 1 \). If \( i > t \), use previous source elements stored in the \( X_0 \) vector.
10. Next \( t \), go to step 3.
11. Store \( N \) ending source elements in previous source elements vector \( X_0 \): if \( N > n, N = n \); else \( N = \text{tap} \).

Each of the FilterInit functions allocates memory for the internal filter structure and converts the parameters into the internal representation.

Each of the Filter functions applies the NLMS adaptive filter on one signal packet and updates the filter states.

Each of the FilterNoAdapt functions applies the NLMS filter on one signal packet and updates the filter states but without changing the filter weights.

Each of the FilterFree functions releases the memory allocated for the internal filter structure.

**Parameters**
Each of the functions takes some of the following arguments:

- **filter** Internal filter structure.
- **flt** Filter coefficient array.
- **tap** Taps of the filter.
- **beta** Error weighting factor. \( 0 < \beta < 1 \).
- **dst** Destination signal array.
- **src** Source signal array.
- **ref** Reference or "desired" signal array.
Number of samples in the source signal array.

Return Values Each of the FilterInit, Filter and FilterNonAdapt functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE. The FilterFree functions don’t return anything.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_SignalLMSFilter(3MLIB), attributes(5)
# mlib_SignalQuant2_S16_F32

## Name
mlib_SignalQuant2_S16_F32 – float to 16-bit quantization

## Synopsis
c
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_SignalQuant2_S16_F32(mlib_s16 *dst,
    const mlib_f32 *src, const mlib_f32 thresh, mlib_s32 length,
    mlib_s16 offset, mlib_s32 n);
```

## Description
The `mlib_SignalQuant2_S16_F32()` function quantizes a signal array by using the following equation:

\[
X = x(n) \quad n = 0, 1, \ldots \\
Z = z(n) \quad n = 0, 1, \ldots \\
= offset \quad \text{for } x(n) < t(0) \\
= offset + k \quad \text{for } t(k) \leq x(n) < t(k+1) \\
= offset + \text{length} - 1 \quad \text{for } x(n) \geq t(\text{length} - 1)
\]

## Parameters
The function takes the following arguments:

- **dst** Output signal array
- **src** Input signal array
- **thresh** Array of thresholds
- **length** Length of the array of thresholds
- **offset** Offset for thresholds
- **n** Number of samples in the input signal array

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
attributes(5)
mlib_SignalQuant2_S16S_F32S() function quantizes a signal array by using the following equation:

\[
X = x(n) \quad n = 0, 1, \ldots \\
Z = z(n) \quad n = 0, 1, \ldots \\
= \text{offset} \quad \text{for } x(n) < t(0) \\
= \text{offset} + k \quad \text{for } t(k) \leq x(n) < t(k+1) \\
= \text{offset} + \text{length} - 1 \quad \text{for } x(n) \geq t(\text{length} - 1)
\]

**Parameters**

- **dst**: Output signal array in two-channel interleaved stereo format.
- **src**: Input signal array in two-channel interleaved stereo format.
- **thresh**: Array of thresholds.
- **length**: Length of the array of thresholds.
- **offset**: Offset for thresholds.
- **n**: Number of samples in the input signal array.

**Return Values**

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

attributes(5)
mlib_SignalQuant_S16_F32 – float to 16-bit quantization

#include <mlib.h>

mlib_status mlib_SignalQuant_S16_F32(mlib_s16 *dst, const mlib_f32 *src, const mlib_f32 *thresh, mlib_s32 n);

The mlib_SignalQuant_S16_F32() function quantizes a signal array by using the following equation:

\[
X = x(n) \quad n = 0, 1, \ldots \\
Z = z(n) \quad n = 0, 1, \ldots \\
  = -32768 \quad \text{for } x(n) < t(-32768) \\
  = k \quad \text{for } t(k) \leq x(n) < t(k+1) \\
  = +32767 \quad \text{for } x(n) \geq t(+32767)
\]

The function takes the following arguments:

- \textit{dst} – Output signal array
- \textit{src} – Input signal array
- \textit{thresh} – Array of 65536 thresholds.
- \textit{n} – Number of samples in the input signal array.

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
The `mlib_SignalQuant_S16S_F32S()` function quantizes a signal array by using the following equation:

\[
X = x(n) \quad n = 0, 1, ... \\
Z = z(n) \quad n = 0, 1, ... \\
\begin{align*}
&= -32768 \quad \text{for } x(n) < t(-32768) \\
&= k \quad \text{for } t(k) \leq x(n) < t(k+1) \\
&= +32767 \quad \text{for } x(n) \geq t(+32767)
\end{align*}
\]

**Parameters**

The function takes the following arguments:

- `dst` Output signal array in two-channel interleaved stereo format.
- `src` Input signal array in two-channel interleaved stereo format.
- `thresh` Array of 65536 thresholds.
- `n` Number of samples in the input signal array.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`attributes(5)`
The `mlib_SignalQuant_U8_F32()` function quantizes a signal array by using the following equation:

\[ X = x(n) \quad n = 0, 1, \ldots \]
\[ Z = z(n) \quad n = 0, 1, \ldots \]
\[ = 0 \quad \text{for } x(n) < t(0) \]
\[ = k \quad \text{for } t(k) \leq x(n) < t(k+1) \]
\[ = 255 \quad \text{for } x(n) \geq t(255) \]

The function takes the following arguments:

- `dst` Output signal array
- `src` Input signal array
- `thresh` Array of 256 thresholds.
- `n` Number of samples in the input signal array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalQuant_U8_F32(mlib_u8 *dst,
    const mlib_f32 *src, const mlib_f32 *thresh, mlib_s32 n);
```
mlib_SignalQuant_U8_S16, mlib_SignalQuant_U8S_S16 - 16-bit to 8-bit quantization

Synopsis
c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalQuant_U8_S16(mlib_u8 *dst,
const mlib_s16 *src,
const mlib_s16 *thresh,
mlib_s32 n);

mlib_status mlib_SignalQuant_U8S_S16S(mlib_u8 *dst,
const mlib_s16 *src,
const mlib_s16 *thresh,
mlib_s32 n);

Description
Each of these functions quantizes a signal array by using the following equation:

X = x(n)  n = 0, 1, ...

Z = z(n)  n = 0, 1, ...

= 0  for x(n) < t(0)
= k  for t(k) ≤ x(n) < t(k+1)
= 255 for x(n) ≥ t(255)

Parameters
Each of the functions takes the following arguments:

dst  Output signal array.
src  Input signal array.
thresh  Array of 256 thresholds.
n  Number of samples in the input signal array.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
# Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalQuant_U8S_F32S(mlib_u8 *dst,
const mlib_f32 *src, const mlib_f32 *thresh, mlib_s32 n);

## Description

The `mlib_SignalQuant_U8S_F32S()` function quantizes a signal array by using the following equation:

\[
X = x(n) \quad n = 0, 1, ... \\
Z = z(n) \quad n = 0, 1, ... \\
= 0 \quad \text{for} \quad x(n) < t(0) \\
= k \quad \text{for} \quad t(k) \leq x(n) < t(k+1) \\
= 255 \quad \text{for} \quad x(n) \geq t(255)
\]

The function takes the following arguments:

- **dst**: Output signal array in two-channel interleaved stereo format.
- **src**: Input signal array in two-channel interleaved stereo format.
- **thresh**: Array of 256 thresholds.
- **n**: Number of samples in the input signal array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
mlib_SignalReSampleFIR_F32_F32 - resampling with filtering

Synopsis

cc [ flag... ] file... -lmllib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalReSampleFIR_F32_F32(mlib_f32 *dst,
const mlib_f32 *src, void *state, mlib_s32 n);

Description
The mlib_SignalReSampleFIR_F32_F32() function performs rational sample rate
conversion with FIR filtering between the upsampling and downsampling.

Parameters
The function takes the following arguments:

dst Output signal array.
src Input signal array.
state Internal state structure.
n Number of samples in the input signal array.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
attributes(5)
mlib_SignalReSampleFIR_F32S_F32S() – resampling with filtering

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalReSampleFIR_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, void *state, mlib_s32 n);

Description The mlib_SignalReSampleFIR_F32S_F32S() function performs rational sample rate conversion with FIR filtering between the upsampling and downsampling.

Parameters The function takes the following arguments:
  
  - **dst**  
    - Output signal array in two-channel interleaved stereo format.
  
  - **src**  
    - Input signal array in two-channel interleaved stereo format.
  
  - **state**  
    - Internal state structure.
  
  - **n**  
    - Number of samples in the input signal array.

Return Values The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
The `mlib_SignalReSampleFIRFree_F32_F32()` function releases the memory allocated for the internal state structure for rational sample rate conversion with FIR filtering between upsampling and downsampling.

**Parameters**

The function takes the following arguments:

- `state`  
  Internal state structure.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

attributes(5)
Name  mlib_SignalReSampleFIRFree_F32S_F32S – resampling with filtering

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalReSampleFIRFree_F32S_F32S(void *state);

Description  The mlib_SignalReSampleFIRFree_F32S_F32S() function releases the memory allocated for the internal state structure for rational sample rate conversion with FIR filtering between upsampling and downsampling.

Parameters  The function takes the following arguments:

state       Internal state structure.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
mlib_SignalReSampleFIRFree_S16_S16, mlib_SignalReSampleFIRFree_S16S_S16S

resampling with filtering

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalReSampleFIRFree_S16_S16(void *state);
void mlib_SignalReSampleFIRFree_S16S_S16S(void *state);

Each of these functions releases the memory allocated for the internal state structure for
rational sample rate conversion with FIR filtering between upsampling and downsampling.

Parameters
Each of the functions takes the following arguments:
state Internal state structure.

Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalReSampleFIR_S16_S16_Sat(3MLIB),
mlib_SignalReSampleFIRInit_S16_S16(3MLIB), attributes(5)
mlib_SignalReSampleFIRInit_S16_S16, mlib_SignalReSampleFIRInit_S16S_S16S, mlib_SignalReSampleFIRInit_F32_F32, mlib_SignalReSampleFIRInit_F32S_F32S – initialization for resampling with filtering

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalReSampleFIRInit_S16_S16(void **state, const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor, mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);
mlib_status mlib_SignalReSampleFIRInit_S16S_S16S(void **state, const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor, mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);
mlib_status mlib_SignalReSampleFIRInit_F32_F32(void **state, const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor, mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);
mlib_status mlib_SignalReSampleFIRInit_F32S_F32S(void **state, const mlib_f32 *flt, mlib_s32 tap, mlib_s32 ufactor, mlib_s32 uphase, mlib_s32 dfactor, mlib_s32 dphase);

Each of these functions allocates memory for the internal state structure and converts the parameters into an internal representation for rational sample rate conversion with FIR filtering between upsampling and downsampling.

Parameters
Each of the functions takes the following arguments:
- state: Internal state structure.
- flt: Filter coefficient array, two-channel interleaved in the cases of stereo.
- tap: Taps of the filter.
- ufactor: Factor by which to upsample.
- uphase: Phase in upsampling. 0 ≤ uphase < ufactor.
- dfactor: Factor by which to downsample.
- dphase: Phase in downsampling. 0 ≤ dphase < dfactor.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

- `mlib_SignalReSampleFIR_Init_S16_S16(3MLIB)`
- `mlib_SignalReSampleFIR_F32_F32(3MLIB)`
- `mlib_SignalReSampleFIRFree_S16_S16(3MLIB)`
- `mlib_SignalReSampleFIRFree_F32_F32(3MLIB)`, `attributes(5)`
mlib_SignalReSampleFIR_S16_S16_Sat, mlib_SignalReSampleFIR_S16S_S16S_Sat - resampling with filtering

Synopsis
c [ flag... ] file... -lm [ lib... ]
#include <mlib.h>

mlib_status mlib_SignalReSampleFIR_S16_S16_Sat(mlib_s16 *dst,
    const mlib_s16 *src, void *state, mlib_s32 n);

mlib_status mlib_SignalReSampleFIR_S16S_S16S_Sat(mlib_s16 *dst,
    const mlib_s16 *src, void *state, mlib_s32 n);

Description Each of these functions performs rational sample rate conversion with FIR filtering between the upsampling and downsampling.

Parameters Each of the functions takes the following arguments:

dst Output signal array.
src Input signal array.
state Internal state structure.
n Number of samples in the input signal array.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
mlib_SignalSineWave_F32 – sine wave generation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalSineWave_F32(mlib_f32 *sine, void *state,
        mlib_s32 n);

Description

The mlib_SignalSineWave_F32() function generates one packet of sine wave and updates the internal state.

Parameters

The function takes the following arguments:

- **sine**: Generated sine wave array.
- **state**: Internal state structure.
- **n**: Length of the generated sine wave array in number of samples.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

attributes(5)
Name: mlib_SignalSineWaveFree_F32 – sine wave generation

Synopsis: cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalSineWaveFree_F32(void *state);

Description: The mlib_SignalSineWaveFree_F32() function releases the memory allocated for the internal state's structure.

Parameters: The function takes the following arguments:

state Internal state structure.

Return Values: None.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_SignalSineWave_S16(3MLIB), mlib_SignalSineWaveInit_S16(3MLIB), attributes(5)
mlib_SignalSineWaveFree_S16 – sine wave generation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalSineWaveFree_S16(void *state);

Description

The mlib_SignalSineWaveFree_S16() function releases the memory allocated for the internal state’s structure.

Parameters

The function takes the following arguments:

state Internal state structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

attributes(5)
mlib_SignalSineWaveInit_F32 – sine wave generation

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalSineWaveInit_F32(void **state,
            mlib_f32  mag, mlib_f32  freq, mlib_f32  phase);

Description

The mlib_SignalSineWaveInit_F32() function allocates memory for an internal state structure and converts the parameters of the wave to an internal representation.

Parameters

The function takes the following arguments:

- **state**  Internal state structure.
- **mag**  Magnitude of sine wave to be generated, in Q15 format.
- **freq**  Angular frequency of the sine wave to be generated, measured in radians per sample.
- **phase**  Start phase of the sine wave to be generated, measured in radians.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

attributes(5)
# mlib_SignalSineWaveInit_S16

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_SignalSineWaveInit_S16(void **state, mlib_s16 mag, mlib_f32 freq, mlib_f32 phase);
```

## Description

The `mlib_SignalSineWaveInit_S16()` function allocates memory for an internal state structure and converts the parameters of the wave to an internal representation.

## Parameters

The function takes the following arguments:

- `state` Internal state structure.
- `mag` Magnitude of sine wave to be generated, in Q15 format.
- `freq` Angular frequency of the sine wave to be generated, measured in radians per sample.
- `phase` Start phase of the sine wave to be generated, measured in radians.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_SignalSineWave_S16(3MLIB)`, `mlib_SignalSineWaveFree_S16(3MLIB)`
- attributes(5)
The `mlib_SignalSineWave_S16()` function generates one packet of sine wave and updates the internal state.

### Parameters
- `sine`  
  Generated sine wave array.
- `state`  
  Internal state structure.
- `n`  
  Length of the generated sine wave array in number of samples.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_SignalSineWave_S16(3MLIB)`, `mlib_SignalSineWaveInit_S16(3MLIB)`, `attributes(5)`
mlib_signal_split_F32_F32S(3MLIB)

Name mlib_SignalSplit_F32_F32S – split

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalSplit_F32_F32S(mlib_f32 *ch0,
               mlib_f32 *ch1, const mlib_f32 *src, mlib_s32 n);

Description The following function splits a stereo signal array into two signal arrays.

Parameters The function takes the following arguments:

- ch0 Destination signal array of Channel 0.
- ch1 Destination signal array of Channel 1.
- n Number of samples in the source signal array.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
# mlib_SignalSplit_S16_S16S

## Description

The following function splits a stereo signal array into two signal arrays.

## Parameters

- **ch0**: Destination signal array of Channel 0.
- **ch1**: Destination signal array of Channel 1.
- **n**: Number of samples in the source signal array.

## Return Values

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

mlib_SignalMerge_S16S_S16(3MLIB), attributes(5)
Name
mlib_SignaluLaw2ALaw – ITU G.711 m-law and A-law compression and decompression

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignaluLaw2ALaw(mlib_u8 *acode,
    const mlib_u8 *ucode, mlib_s32 n);

Description
The mlib_SignaluLaw2ALaw() function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (formerly CCITT) G.711 specification.

Parameters
The function takes the following arguments:

- acode: A-law code array.
- ucode: m-law code array.
- n: Number of samples in the input array.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalALaw2Linear(3MLIB), mlib_SignalALaw2uLaw(3MLIB),
mlib_SignalALaw2uLaw2ALaw(3MLIB), mlib_SignalLinear2uLaw(3MLIB),
mlib_SignalLinear2ALaw(3MLIB), mlib_SignaluLaw2ALaw(3MLIB), attributes(5)
mlib_SignaluLaw2Linear – ITU G.711 m-law and A-law compression and decompression

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignaluLaw2Linear(mlib_s16 *pcm,
                           const mlib_u8 *ucode, mlib_s32 n);
```

Description

The `mlib_SignaluLaw2Linear()` function performs ITU G.711 m-law and A-law compression and decompression in compliance with the ITU (formerly CCITT) G.711 specification.

Parameters

- `pcm` - Linear PCM sample array.
- `ucode` - m-law code array.
- `n` - Number of samples in the input array.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_SignaluLaw2Linear(3MLIB)`, `mlib_SignaluLaw2uLaw(3MLIB)`, `mlib_SignalLinear2ALaw(3MLIB)`, `mlib_SignalLinear2uLaw(3MLIB)`, `mlib_SignaluLaw2ALaw(3MLIB)`, `attributes(5)`
The `mlib_SignalUpSampleFIR_F32_F32` function performs upsampling immediately followed by FIR filtering on one packet of signal and updates the internal state. The function takes the following arguments:

- `dst` Output signal array.
- `src` Input signal array.
- `state` Internal state structure.
- `n` Number of samples in the input signal array.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

`attributes(5)`
mlib_SignalUpSampleFIR_F32S_F32S – upsampling with filtering

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalUpSampleFIR_F32S_F32S(mlib_f32 *dst,
                                           const mlib_f32 *src, void *state, mlib_s32 n);
```

Description

The `mlib_SignalUpSampleFIR_F32S_F32S()` function performs upsampling immediately followed by FIR filtering on one packet of signal and updates the internal state.

Parameters

The function takes the following arguments:

- `state`: Internal state structure.
- `n`: Number of samples in the input signal array.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`attributes(5)`
The `mlib_SignalUpSampleFIRFree_F32_F32()` function releases the memory allocated for the internal state structure for upsampling immediately followed by FIR filtering.

The function takes the following arguments:

```
state    Internal state structure.
```

Return Values
None.

Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
`attributes(5)`
mlib_SignalUpSampleFIRFree_F32S_F32S – upsampling with filtering

#include <mlib.h>

void mlib_SignalUpSampleFIRFree_F32S_F32S(void *state);

The mlib_SignalUpSampleFIRFree_F32S_F32S() function releases the memory allocated for
the internal state structure for upsampling immediately followed by FIR filtering.

Parameters

state Internal state structure.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
mlib_SignalUpSampleFIRFree_S16_S16

Name
mlib_SignalUpSampleFIRFree_S16_S16, mlib_SignalUpSampleFIRFree_S16S_S16S – upsampling with filtering

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_SignalUpSampleFIRFree_S16_S16(void *state);
void mlib_SignalUpSampleFIRFree_S16S_S16S(void *state);

Description
Each of these functions releases the memory allocated for the internal state structure for upsampling immediately followed by FIR filtering.

Parameters
Each of the functions takes the following arguments:

state Internal state structure.

Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_SignalUpSampleFIR_S16_S16_Sat(3MLIB),
mlib_SignalUpSampleFIRInit_S16_S16(3MLIB), attributes(5)
### Name
mlib_SignalUpSampleFIRInit_F32_F32 – upsampling with filtering

### Synopsis
```c
cc [ ... ] file... -lmlib [ ... ]
#include <mlib.h>

mlib_status mlib_SignalUpSampleFIRInit_F32_F32(void **state,
                             const mlib_f32 *flt, mlib_s32 tap, mlib_s32 factor,
                             mlib_s32 phase);
```

### Description
The `mlib_SignalUpSampleFIRInit_F32_F32()` function allocates memory for the internal state structure and converts the parameters into an internal representation for upsampling immediately followed by FIR filtering.

### Parameters
The function takes the following arguments:
- `state` Internal state structure.
- `flt` Filter coefficient array.
- `tap` Taps of the filter.
- `factor` Factor by which to upsample.
- `phase` Parameter that determines the relative position of an input value, within the output signal. $0 \leq \text{phase} < \text{factor}$.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)
The `mlib_SignalUpSampleFIRInit_F32S_F32S()` function allocates memory for the internal state structure and converts the parameters into an internal representation for upsampling immediately followed by FIR filtering.

### Parameters
The function takes the following arguments:

- **state**: Internal state structure.
- **flt**: Filter coefficient array in two-channel stereo format. src[2*i] contains channel 0, and src[2*i+1] contains channel 1 array.
- **tap**: Taps of the filter.
- **factor**: Factor by which to upsample.
- **phase**: Parameter that determines the relative position of an input value, within the output signal. 0 ≤ phase < factor.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
`attributes(5)`
mlib_SignalUpSampleFIRInit_S16_S16(3MLIB)

Name  
mlib_SignalUpSampleFIRInit_S16_S16, mlib_SignalUpSampleFIRInit_S16S_S16S

Synopsis  
c [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalUpSampleFIRInit_S16_S16(void **state, const mlib_f32 *flt, mlib_s32 tap, mlib_s32 factor, mlib_s32 phase);
mlib_status mlib_SignalUpSampleFIRInit_S16S_S16S(void **state, const mlib_f32 *flt, mlib_s32 tap, mlib_s32 factor, mlib_s32 phase);

Description  
Each of these functions allocates memory for the internal state structure and converts the parameters into an internal representation for upsampling immediately followed by FIR filtering.

Parameters  
Each of the functions takes the following arguments:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Internal state structure.</td>
</tr>
<tr>
<td>flt</td>
<td>Filter coefficient array.</td>
</tr>
<tr>
<td>tap</td>
<td>Taps of the filter.</td>
</tr>
<tr>
<td>factor</td>
<td>Factor by which to upsample.</td>
</tr>
<tr>
<td>phase</td>
<td>Parameter that determines the relative position of an input value, within the output signal. $0 \leq \text{phase} &lt; \text{factor}$.</td>
</tr>
</tbody>
</table>

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_SignalUpSampleFIR_S16_S16_Sat(3MLIB),
mlib_SignalUpSampleFIRFree_S16_S16(3MLIB), attributes(5)
**Name**
mlib_SignalUpSampleFIR_S16_S16_Sat, mlib_SignalUpSampleFIR_S16S_S16S_Sat – upsampling with filtering

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```
mlib_status mlib_SignalUpSampleFIR_S16_S16_Sat(mlib_s16 *dst,
       const mlib_s16 *src, void *state, mlib_s32 n);
mlib_status mlib_SignalUpSampleFIR_S16S_S16S_Sat(mlib_s16 *dst,
       const mlib_s16 *src, void *state, mlib_s32 n);
```

**Description**
Each of these functions performs upsampling immediately followed by FIR filtering on one packet of signal and updates the internal state.

**Parameters**
Each of the functions takes the following arguments:
- `dst` Output signal array.
- `src` Input signal array.
- `state` Internal state structure.
- `n` Number of samples in the input signal array.

**Return Values**
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_SignalUpSampleFIRFree_S16_S16(3MLIB),
mlib_SignalUpSampleFIRInit_S16_S16(3MLIB), attributes(5)
mlib_SignalUpSample_S16_S16, mlib_SignalUpSample_S16S_S16S, mlib_SignalUpSample_F32_F32, mlib_SignalUpSample_F32S_F32S – signal upsampling

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalUpSample_S16_S16(mlib_s16 *dst,
    const mlib_s16 *src, mlib_s32 factor, mlib_s32 phase,
    mlib_s32 n);

mlib_status mlib_SignalUpSample_S16S_S16S(mlib_s16 *dst,
    const mlib_s16 *src, mlib_s32 factor, mlib_s32 phase,
    mlib_s32 n);

mlib_status mlib_SignalUpSample_F32_F32(mlib_f32 *dst,
    const mlib_f32 *src, mlib_s32 factor, mlib_s32 phase,
    mlib_s32 n);

mlib_status mlib_SignalUpSample_F32S_F32S(mlib_f32 *dst,
    const mlib_f32 *src, mlib_s32 factor, mlib_s32 phase,
    mlib_s32 n);

Description

Each of these functions performs upsampling.

For monaural signals, the following equation is used:

dst[i] = src[k] if i == k*factor + phase
dst[i] = 0 if i != k*factor + phase

where k = 0, 1, ..., (n - 1); i = 0, 1, ..., (n*factor - 1).

For stereo signals, the following equation is used:

dst[2*i] = src[2*k] if i == k*factor + phase
dst[2*i] = 0 if i != k*factor + phase

dst[2*i + 1] = src[2*k + 1] if i == k*factor + phase
dst[2*i + 1] = 0 if i != k*factor + phase

where k = 0, 1, ..., (n - 1); i = 0, 1, ..., (n*factor - 1).

Parameters

Each of the functions takes the following arguments:

dst Output signal array.
src Input signal array.
factor Factor by which to upsample. factor ≥ 1.
phase Parameter that determines relative position of an input value, within the output signal. 0 ≤ phase < factor.
n Number of samples in the input signal array.
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_SignalDownSample_S16_S16(3MLIB), attributes(5)
mlib_SignalWhiteNoise_F32 — white noise generation

Synopsis

```c
cc [ flag... ] file... -lmllib [ library... ]
#include <mlib.h>

mlib_status mlib_SignalWhiteNoise_F32(mlib_f32 *wnoise,
   void *state, mlib_s32 n);
```

Description
The `mlib_SignalWhiteNoise_F32()` function generates one packet of white noise and updates the internal state.

Parameters
The function takes the following arguments:
- `wnoise`: Generated white noise array.
- `state`: Internal state structure.
- `n`: Length of the generated sine wave array in number of samples.

Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
`attributes(5)`
The `mlib_SignalWhiteNoiseFree_F32()` function releases the memory allocated for the internal state’s structure.

Parameters  
The function takes the following arguments:

- `state` Internal state structure.

Attributes  
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
`attributes(5)`
mlib_SignalWhiteNoiseFree_S16 – white noise generation

Synopsis

`cc [ flag... ] file... -lmlib [ library... ]`

```c
#include <mlib.h>

void mlib_SignalWhiteNoiseFree_S16(void *state);
```

Description

The `mlib_SignalWhiteNoiseFree_S16()` function releases the memory allocated for the internal state’s structure.

Parameters

The function takes the following arguments:

- `state` Internal state structure.

Return Values

None.

Attributes

See [attributes](5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_SignalWhiteNoise_S16(3MLIB)`, `mlib_SignalWhiteNoiseInit_S16(3MLIB)`, [attributes](5)`
mlib_SignalWhiteNoiseInit_F32 – white noise generation

Synopsis

```c
#include <mlib.h>

mlib_status mlib_SignalWhiteNoiseInit_F32(void **state, mlib_f32 mag,
                                      mlib_f32 seed);
```

Description

The `mlib_SignalWhiteNoiseInit_F32` function allocates memory for an internal state structure and converts the parameters into an internal representation.

Parameters

The function takes the following arguments:

- `state` Internal state structure.
- `mag` Magnitude of white noise to be generated, in Q15 format.
- `seed` Seed value for the pseudorandom number generator.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`attributes(5)`
The `mlib_SignalWhiteNoiseInit_S16()` function allocates memory for an internal state structure and converts the parameters into an internal representation.

### Parameters

- **state**  
  Internal state structure.
- **mag**  
  Magnitude of white noise to be generated, in Q15 format.
- **seed**  
  Seed value for the pseudorandom number generator.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_SignalWhiteNoise_S16(3MLIB)`
- `mlib_SignalWhiteNoiseFree_S16(3MLIB)`
- `attributes(5)`
The `mlib_SignalWhiteNoise_S16()` function generates one packet of white noise and updates the internal state.

The function takes the following arguments:

- `wnoise`: Generated white noise array.
- `state`: Internal state structure.
- `n`: Length of the generated sine wave array in number of samples.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_SignalWhiteNoiseFree_S16(3MLIB)`, `mlib_SignalWhiteNoiseInit_S16(3MLIB)`, attributes(5)
Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorAddS_U8_Mod(mlib_u8 *xz,
         const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_U8_Sat(mlib_u8 *xz,
         const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_U8C_Mod(mlib_u8 *xz,
         const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_U8C_Sat(mlib_u8 *xz,
         const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8_Mod(mlib_s8 *xz,
         const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8_Sat(mlib_s8 *xz,
         const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8C_Mod(mlib_s8 *xz,
         const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8C_Sat(mlib_s8 *xz,
         const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16_Mod(mlib_s16 *xz,
         const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16_Sat(mlib_s16 *xz,
         const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_Mod(mlib_s16 *xz,
         const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_Sat(mlib_s16 *xz,
         const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32_Mod(mlib_s32 *xz,
         const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32_Sat(mlib_s32 *xz,
         const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_Mod(mlib_s32 *xz,
         const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_Sat(mlib_s32 *xz,
         const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n);

Description Each of these functions performs an in-place addition of a scalar to a vector.

For real data, the following equation is used:
    \[ x[z][i] = c[0] + x[z][i] \]
where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:
    \[
    \begin{align*}
    x[z][2*i] &= c[0] + x[z][2*i] \\
    x[z][2*i + 1] &= c[1] + x[z][2*i + 1]
    \end{align*}
    \]
where \( i = 0, 1, \ldots, (n - 1) \).

Parameters Each of the functions takes the following arguments:
    
xz Pointer to the first element of the source and destination vector.
    
c Pointer to the source scalar. When the function is used with complex data types, \( c[0] \)
        contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
    
n Number of elements in the vectors.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VectorAddS_U8_U8_Mod(3MLIB), attributes(5)
**Name**

- mlib_VectorAddS_U8_U8_Mod
- mlib_VectorAddS_U8_U8_Sat
- mlib_VectorAddS_U8C_U8C_Mod
- mlib_VectorAddS_U8C_U8C_Sat
- mlib_VectorAddS_S8_S8_Mod
- mlib_VectorAddS_S8_S8_Sat
- mlib_VectorAddS_S8C_S8C_Mod
- mlib_VectorAddS_S8C_S8C_Sat
- mlib_VectorAddS_S16_U8_Mod
- mlib_VectorAddS_S16_U8_Sat
- mlib_VectorAddS_S16_S8_Mod
- mlib_VectorAddS_S16_S8_Sat
- mlib_VectorAddS_S16_S16_Mod
- mlib_VectorAddS_S16_S16_Sat
- mlib_VectorAddS_S16C_U8C_Mod
- mlib_VectorAddS_S16C_U8C_Sat
- mlib_VectorAddS_S16C_S8C_Mod
- mlib_VectorAddS_S16C_S8C_Sat
- mlib_VectorAddS_S16C_S16C_Mod
- mlib_VectorAddS_S16C_S16C_Sat
- mlib_VectorAddS_S32_S16_Mod
- mlib_VectorAddS_S32_S16_Sat
- mlib_VectorAddS_S32_S32_Mod
- mlib_VectorAddS_S32_S32_Sat
- mlib_VectorAddS_S32C_S16C_Mod
- mlib_VectorAddS_S32C_S16C_Sat
- mlib_VectorAddS_S32C_S32C_Mod
- mlib_VectorAddS_S32C_S32C_Sat

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorAddS_U8_U8_Mod(mlib_u8 *z,
   const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_U8_U8_Sat(mlib_u8 *z,
   const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_U8C_U8C_Mod(mlib_u8 *z,
   const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_U8C_U8C_Sat(mlib_u8 *z,
   const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8_S8_Mod(mlib_s8 *z,
   const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8_S8_Sat(mlib_s8 *z,
   const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8C_S8C_Mod(mlib_s8 *z,
   const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S8C_S8C_Sat(mlib_s8 *z,
   const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16_S8_Mod(mlib_s16 *z,
   const mlib_s16 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16_S8_Sat(mlib_s16 *z,
   const mlib_s16 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_S16C_Mod(mlib_s16 *z,
   const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_S16C_Sat(mlib_s16 *z,
   const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
```
mlib_status mlib_VectorAddS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorAddS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

Description  Each of these functions adds a scalar to a vector.

For real data, the following equation is used:

\[ z[i] = c[0] + x[i] \]
where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
z_{2i} &= c[0] + x_{2i} \\
z_{2i + 1} &= c[1] + x_{2i + 1}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the destination vector.
- \( x \) Pointer to the first element of the source vector.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( n \) Number of elements in the vectors.

**Return Values** Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise, it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** `mlib_VectorAddS_U8_U8_Mod(3MLIB)`, attributes(5)
Name  
mlib_VectorAdd_U8_Mod, mlib_VectorAdd_U8_Sat, mlib_VectorAdd_U8C_Mod,
mlib_VectorAdd_U8C_Sat, mlib_VectorAdd_S8_Mod, mlib_VectorAdd_S8_Sat,
mlib_VectorAdd_S8C_Mod, mlib_VectorAdd_S8C_Sat, mlib_VectorAdd_S16_Mod,
mlib_VectorAdd_S16_Sat, mlib_VectorAdd_S16C_Mod, mlib_VectorAdd_S16C_Sat,
mlib_VectorAdd_S32_Mod, mlib_VectorAdd_S32_Sat, mlib_VectorAdd_S32C_Mod,
mlib_VectorAdd_S32C_Sat

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorAdd_U8_Mod(mlib_u8 *xz,
const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_U8_Sat(mlib_u8 *xz,
const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_U8C_Mod(mlib_u8 *xz,
const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_U8C_Sat(mlib_u8 *xz,
const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8_Mod(mlib_s8 *xz,
const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8_Sat(mlib_s8 *xz,
const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8C_Mod(mlib_s8 *xz,
const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8C_Sat(mlib_s8 *xz,
const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_Mod(mlib_s16 *xz,
const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_Sat(mlib_s16 *xz,
const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_Mod(mlib_s16 *xz,
const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_Sat(mlib_s16 *xz,
const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32_Mod(mlib_s32 *xz,
const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32_Sat(mlib_s32 *xz,
const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32C_Mod(mlib_s32 *xz,
const mlib_s32 *y, mlib_s32 n);
MLib status mlib_VectorAdd_S32C_Sat(mlib_s32 *xz,  
    const mlib_s32 *y, mlib_s32 n);

Description Each of these functions performs the in-place addition of one vector to another vector. It uses the following equation:

\[ xz[i] = xz[i] + y[i] \]

where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2n - 1) \) for complex data.

Parameters Each of the functions takes the following arguments:

- \( xz \) Pointer to the first element of the first source and destination vector.
- \( y \) Pointer to the first element of the second source vector.
- \( n \) Number of elements in the vectors.

Return Values Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib.VectorAdd_U8_U8_Mod(3MLIB), attributes(5)
Name  
mlib_VectorAdd_U8_U8_Mod, mlib_VectorAdd_U8_U8_Sat,
mlib_VectorAdd_U8C_U8C_Mod, mlib_VectorAdd_U8C_U8C_Sat,
mlib_VectorAdd_S8_S8_Mod, mlib_VectorAdd_S8_S8_Sat,
mlib_VectorAdd_S8C_S8C_Mod, mlib_VectorAdd_S8C_S8C_Sat,
mlib_VectorAdd_S16_U8_Mod, mlib_VectorAdd_S16_U8_Sat,
mlib_VectorAdd_S16_S8_Mod, mlib_VectorAdd_S16_S8_Sat,
mlib_VectorAdd_S16_S16_Mod, mlib_VectorAdd_S16_S16_Sat,
mlib_VectorAdd_S16C_U8C_Mod, mlib_VectorAdd_S16C_U8C_Sat,
mlib_VectorAdd_S16C_S8C_Mod, mlib_VectorAdd_S16C_S8C_Sat,
mlib_VectorAdd_S16C_S16C_Mod, mlib_VectorAdd_S16C_S16C_Sat,
mlib_VectorAdd_S32_S16_Mod, mlib_VectorAdd_S32_S16_Sat,
mlib_VectorAdd_S32_S32_Mod, mlib_VectorAdd_S32_S32_Sat,
mlib_VectorAdd_S32C_S16C_Mod, mlib_VectorAdd_S32C_S16C_Sat,
mlib_VectorAdd_S32C_S32C_Mod, mlib_VectorAdd_S32C_S32C_Sat – vector addition

Synopsis  
cc [flag...] file... -lmlib [library...]
#include <mlib.h>

mlib_status mlib_VectorAdd_U8_U8_Mod(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_U8_U8_Sat(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_U8C_U8C_Mod(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_U8C_U8C_Sat(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8_S8_Mod(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8_S8_Sat(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8C_S8C_Mod(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S8C_S8C_Sat(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_U8_Mod(mlib_s16 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_U8_Sat(mlib_s16 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_S8_Mod(mlib_s16 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_S8_Sat(mlib_s16 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_S16_Mod(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16_S16_Sat(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_U8C_Mod(mlib_s16c *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_U8C_Sat(mlib_s16c *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_S8C_Mod(mlib_s16c *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_S8C_Sat(mlib_s16c *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_S16C_Mod(mlib_s16c *z,
const mlib_s16c *x, const mlib_s16c *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S16C_S16C_Sat(mlib_s16c *z,
const mlib_s16c *x, const mlib_s16c *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32_S16_Mod(mlib_s32 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32_S16_Sat(mlib_s32 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32_S32_Mod(mlib_s32 *z,
const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32_S32_Sat(mlib_s32 *z,
const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32C_S16C_Mod(mlib_s32c *z,
const mlib_s16c *x, const mlib_s16c *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32C_S16C_Sat(mlib_s32c *z,
const mlib_s16c *x, const mlib_s16c *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32C_S32C_Mod(mlib_s32c *z,
const mlib_s32c *x, const mlib_s32c *y, mlib_s32 n);
mlib_status mlib_VectorAdd_S32C_S32C_Sat(mlib_s32c *z,
const mlib_s32c *x, const mlib_s32c *y, mlib_s32 n);
Each of these functions performs the addition of one vector to another vector. It uses the following equation:

\[ z[i] = x[i] + y[i] \]

where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2n - 1) \) for complex data.

**Parameters** Each of the functions takes the following arguments:
Point to the first element of the destination vector.
Point to the first element of the first source vector.
Point to the first element of the second source vector.
Number of elements in the vectors.

**Return Values**  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_VectorAdd_U8_U8_Mod(3MLIB), attributes(5)
mlib_VectorAng_U8C, mlib_VectorAng_S8C, mlib_VectorAng_S16C, mlib_VectorAng_S32C – vector complex phase (angle)

Synopsis

```c
cc [ flag... ] file... -lmllib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorAng_U8C(mlib_d64 *a,
    const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorAng_S8C(mlib_d64 *a,
    const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorAng_S16C(mlib_d64 *a,
    const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorAng_S32C(mlib_d64 *a,
    const mlib_s32 *x, mlib_s32 n);
```

Description

Each of these functions computes the phase vector of a complex vector.

The following equation is used:

\[ a[i] = \text{atan}(x[2*i + 1] / x[2*i]) \]

where \( i = 0, 1, \ldots, (n - 1) \).

Parameters

Each of the functions takes the following arguments:

- **a** Pointer to the destination phase vector.
- **x** Pointer to the source vector
- **n** Number of elements in the vector.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

attributes(5)
Each of these functions performs an in-place averaging of two vectors. It uses the following equation:

\[ x_{z}[i] = \frac{x_{z}[i] + y[i] + 1}{2} \]

where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2*n - 1) \) for complex data.

**Parameters**

Each of the functions takes the following arguments:

- \( xz \)  
  Pointer to the first element of the first source and destination vector.

- \( y \)  
  Pointer to the first element of the second source vector.

- \( n \)  
  Number of elements in the vectors.

**Return Values**

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_VectorAve_U8(3MLIB), attributes(5)`
Each of these functions computes the average of two vectors. It uses the following equation:

Description

### Synopsis

cc [ flag... ] file... -lmlib [ library... ]

```
#include <mlib.h>

mlib_status mlib_VectorAve_U8_U8(mlib_u8 *z,  
        const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_U8C_U8C(mlib_u8 *z,  
        const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S8_S8(mlib_s8 *z,  
        const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S8C_S8C(mlib_s8 *z,  
        const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S16_U8(mlib_s16 *z,  
        const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S16_S8(mlib_s16 *z,  
        const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S16_S16(mlib_s16 *z,  
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S16C_U8C(mlib_s16 *z,  
        const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S16C_S8C(mlib_s16 *z,  
        const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S16C_S16C(mlib_s16 *z,  
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S32_S16(mlib_s32 *z,  
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S32_S32(mlib_s32 *z,  
        const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S32C_S16C(mlib_s32 *z,  
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorAve_S32C_S32C(mlib_s32 *z,  
        const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
```
\[ z[i] = \frac{x[i] + y[i] + 1}{2} \]

where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2n - 1) \) for complex data.

**Parameters** Each of the functions takes the following arguments:
- \( z \) Pointer to the first element of the destination vector.
- \( x \) Pointer to the first element of the first source vector.
- \( y \) Pointer to the first element of the second source vector.
- \( n \) Number of elements in the vectors.

**Return Values** Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

**Attributes** See attributes\(^{(5)}\) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib\_VectorAve\_U8(3MLIB), attributes\(^{(5)}\)
Name  mlib_VectorConjRev_S8C_S8C_Sat, mlib_VectorConjRev_S16C_S16C_Sat, mlib_VectorConjRev_S32C_S32C_Sat – vector conjugation reversion

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorConjRev_S8C_S8C_Sat(mlib_s8 *z,
const mlib_s8 *x, mlib_s32 n);

mlib_status mlib_VectorConjRev_S16C_S16C_Sat(mlib_s16 *z,
const mlib_s16 *x, mlib_s32 n);

mlib_status mlib_VectorConjRev_S32C_S32C_Sat(mlib_s32 *z,
const mlib_s32 *x, mlib_s32 n);

Description  Each of these functions computes the complex reversion of a complex vector.

The source and destination vectors must be in the same data type.

The following equation is used:

\[
\begin{align*}
z[2*i] & = x[2*(n-1-i)] \\
z[2*i + 1] & = -x[2*(n-1-i) + 1]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

Parameters  Each of the functions takes the following arguments:

- \( z \)  Pointer to the first element of the destination vector.
- \( x \)  Pointer to the first element of the source vector.
- \( n \)  Number of elements in the vectors.

Return Values  Each of the functions returns MMLIB_SUCCESS if successful. Otherwise it returns MMLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
Name  
mlib_VectorConj_S8C_S8C_Sat, mlib_VectorConj_S16C_S16C_Sat,  
mlib_VectorConj_S32C_S32C_Sat – vector conjugation

Synopsis  
cc [ flag... ] file... -mlib [ library... ]  
#include <mlib.h>  
mlib_status mlib_VectorConj_S8C_S8C_Sat(mlib_s8 *z,  
const mlib_s8 *x, mlib_s32 n);  
mlib_status mlib_VectorConj_S16C_S16C_Sat(mlib_s16 *z,  
const mlib_s16 *x, mlib_s32 n);  
mlib_status mlib_VectorConj_S32C_S32C_Sat(mlib_s32 *z,  
const mlib_s32 *x, mlib_s32 n);

Description  
Each of these functions computes the complex conjugate of a complex vector.
The source and destination vectors must be in the same data type.
The following equation is used:
\[
z[2i] = x[2i]  
z[2i + 1] = -x[2i + 1]
\]
where \(i = 0, 1, \ldots, (n - 1)\).

Parameters  
Each of the functions takes the following arguments:
\(z\)  
Pointer to the first element of the destination vector.
\(x\)  
Pointer to the first element of the source vector.
\(n\)  
Number of elements in the vectors.

Return Values  
Each of the functions returns MLib_SUCCESS if successful. Otherwise it returns MLib_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
attributes(5)
### mlib_VectorConj_S8C_Sat(3MLIB)

**Name**  
mlib_VectorConj_S8C_Sat, mlib_VectorConj_S16C_Sat, mlib_VectorConj_S32C_Sat – vector conjugation

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

```c
mlib_status mlib_VectorConj_S8C_Sat(mlib_s8 *xz, mlib_s32 n);
mlib_status mlib_VectorConj_S16C_Sat(mlib_s16 *xz, mlib_s32 n);
mlib_status mlib_VectorConj_S32C_Sat(mlib_s32 *xz, mlib_s32 n);
```

**Description**  
Each of these functions computes the in-place complex conjugate of a complex vector.  
The following equation is used:  
xz[2i + 1] = -xz[2i + 1]  
where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**  
Each of the functions takes the following arguments:  
\( xz \) – Pointer to the first element of the source and destination vector.  
\( n \) – Number of elements in the vector.

**Return Values**  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
attributes(5)
Each of these functions computes the complex conjugate extension of a complex vector.

When \( n \) is even, the following equation is used:

\[
\begin{align*}
z[2*i] & = x[2*i] \\
z[2*i + 1] & = -x[2*i + 1]
\end{align*}
\]

for \( i = 0, 1, \ldots, (n - 1) \).

\[
\begin{align*}
z[2*i] & = x[2*(2*n - 1 - i)] \\
z[2*i + 1] & = -x[2*(2*n - 1 - i) + 1]
\end{align*}
\]

for \( i = n, (n + 1), \ldots, (2*n - 1) \).

When \( n \) is odd, the following equation is used:

\[
\begin{align*}
z[2*i] & = x[2*i] \\
z[2*i + 1] & = -x[2*i + 1]
\end{align*}
\]

for \( i = 0, 1, \ldots, (n - 1) \).

\[
\begin{align*}
z[2*i] & = x[2*(2*n - 2 - i)] \\
z[2*i + 1] & = -x[2*(2*n - 2 - i) + 1]
\end{align*}
\]

for \( i = n, (n + 1), \ldots, (2*n - 2) \).

Each of the functions takes the following arguments:

\begin{itemize}
  \item \( z \) Pointer to the first element of the destination vector.
  \item \( x \) Pointer to the first element of the source vector.
  \item \( n \) Number of elements in the source vector.
\end{itemize}
Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
REFERENCE

Multimedia Library Functions - Part 6
Name  
mlib_VectorConvert_U8_S8_Mod, mlib_VectorConvert_U8_S16_Mod, 
mlib_VectorConvert_U8_S32_Mod, mlib_VectorConvert_S8_U8_Mod, 
mlib_VectorConvert_S8_S16_Mod, mlib_VectorConvert_S8_S32_Mod, 
mlib_VectorConvert_S16_U8_Mod, mlib_VectorConvert_S16_S8_Mod, 
mlib_VectorConvert_S16_S32_Mod, mlib_VectorConvert_S32_U8_Mod, 
mlib_VectorConvert_S32_S8_Mod, mlib_VectorConvert_S32_S16_Mod, 
mlib_VectorConvert_U8C_S8C_Mod, mlib_VectorConvert_U8C_S16C_Mod, 
mlib_VectorConvert_U8C_S32C_Mod, mlib_VectorConvert_S8C_U8C_Mod, 
mlib_VectorConvert_S8C_S16C_Mod, mlib_VectorConvert_S8C_S32C_Mod, 
mlib_VectorConvert_S16C_U8C_Mod, mlib_VectorConvert_S16C_S8C_Mod, 
mlib_VectorConvert_S16C_S32C_Mod, mlib_VectorConvert_S32C_U8C_Mod, 
mlib_VectorConvert_S32C_S8C_Mod, mlib_VectorConvert_S32C_S16C_Mod, 
mlib_VectorConvert_U8_S8_Sat, mlib_VectorConvert_U8_S16_Sat, 
mlib_VectorConvert_U8_S32_Sat, mlib_VectorConvert_S8_U8_Sat, 
mlib_VectorConvert_S8_S16_Sat, mlib_VectorConvert_S8_S32_Sat, 
mlib_VectorConvert_S16_U8_Sat, mlib_VectorConvert_S16_S8_Sat, 
mlib_VectorConvert_S16_S32_Sat, mlib_VectorConvert_S32_U8_Sat, 
mlib_VectorConvert_S32_S8_Sat, mlib_VectorConvert_S32_S16_Sat, 
mlib_VectorConvert_U8C_S8C_Sat, mlib_VectorConvert_U8C_S16C_Sat, 
mlib_VectorConvert_U8C_S32C_Sat, mlib_VectorConvert_S8C_U8C_Sat, 
mlib_VectorConvert_S8C_S16C_Sat, mlib_VectorConvert_S8C_S32C_Sat, 
mlib_VectorConvert_S16C_U8C_Sat, mlib_VectorConvert_S16C_S8C_Sat, 
mlib_VectorConvert_S16C_S32C_Sat, mlib_VectorConvert_S32C_U8C_Sat, 
mlib_VectorConvert_S32C_S8C_Sat, mlib_VectorConvert_S32C_S16C_Sat – vector data type convert

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorConvert_U8_S8_Mod(mlib_u8 *z, const mlib_s8 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_U8_S16_Mod(mlib_u8 *z, const mlib_s16 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_U8_S32_Mod(mlib_u8 *z, const mlib_s32 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S8_U8_Mod(mlib_s8 *z, const mlib_u8 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S8_S16_Mod(mlib_s8 *z, const mlib_s16 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S8_S32_Mod(mlib_s8 *z, const mlib_s32 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S16_S32_Mod(mlib_s16 *z, const mlib_s32 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S32_U8_Mod(mlib_s32 *z, const mlib_u8 *x, 
mlib_s32 n);
mlib_status mlib_VectorConvert_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16_S32_Mod(mlib_s16 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32_U8_Mod(mlib_s32 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32_S8_Mod(mlib_s32 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8C_S8C_Mod(mlib_u8 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8C_S16C_Mod(mlib_u8 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8C_S32C_Mod(mlib_u8 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8C_U8C_Mod(mlib_s8 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8C_S16C_Mod(mlib_s8 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8C_S32C_Mod(mlib_s8 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16C_S32C_Mod(mlib_s16 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32C_U8C_Mod(mlib_s32 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32C_S8C_Mod(mlib_s32 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8_S8_Sat(mlib_u8 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8_S16_Sat(mlib_u8 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8_S32_Sat(mlib_u8 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8_U8_Sat(mlib_s8 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8_S16_Sat(mlib_s8 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8_S32_Sat(mlib_s8 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16_S32_Sat(mlib_s16 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32_U8_Sat(mlib_s32 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32_S8_Sat(mlib_s32 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8C_S8C_Sat(mlib_u8 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8C_S16C_Sat(mlib_u8 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_U8C_S32C_Sat(mlib_u8 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8C_U8C_Sat(mlib_s8 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8C_S16C_Sat(mlib_s8 *z, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S8C_S32C_Sat(mlib_s8 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S16C_S32C_Sat(mlib_s16 *z, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorConvert_S32C_U8C_Sat(mlib_s32 *z, const mlib_u8 *x, mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_S8C_Sat(mlib_s32 *z, const mlib_s8 *x, mlib_s32 n);

mlib_status mlib_VectorConvert_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, mlib_s32 n);

Description Each of these functions copies data from one vector to another vector, of different data types.

For real data, the following equation is used:

\[ z[i] = x[i] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
z[2i] &= x[2i] \\
z[2i + 1] &= x[2i + 1]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

See the following tables for available variations of the data type convert function.

<table>
<thead>
<tr>
<th>Type [*]</th>
<th>U8</th>
<th>S8</th>
<th>S16</th>
<th>S32</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S8</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S16</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>S32</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type [*]</th>
<th>U8C</th>
<th>S8C</th>
<th>S16C</th>
<th>S32C</th>
</tr>
</thead>
<tbody>
<tr>
<td>U8C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S8C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S16C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>S32C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

[*] Each row represents a source data type. Each column represents a destination data type.
Parameters  Each of the functions takes the following arguments:

- $z$  Pointer to the first element of the destination vector.
- $x$  Pointer to the first element of the source vector.
- $n$  Number of elements in the vectors.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTETYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
Each of these functions copies one vector to another vector of the same data type. The input and output vectors must be in the same data type.

For real data, the following equation is used:

\[ z[i] = x[i] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[ z[2i] = x[2i] \]
\[ z[2i + 1] = x[2i + 1] \]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the destination vector.
- \( x \) Pointer to the first element of the source vector.
- \( n \) Number of elements in the vectors.
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
attributes(5)
Name  mlib_VectorDistance_U8_Sat, mlib_VectorDistance_S8_Sat, mlib_VectorDistance_S16_Sat, mlib_VectorDistance_S32_Sat – vector Euclidean distance

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorDistance_U8_Sat(mlib_d64 *z, const mlib_u8 *x,
           const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorDistance_S8_Sat(mlib_d64 *z, const mlib_s8 *x,
           const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorDistance_S16_Sat(mlib_d64 *z, const mlib_s16 *x,
           const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorDistance_S32_Sat(mlib_d64 *z, const mlib_s32 *x,
           const mlib_s32 *y, mlib_s32 n);

Description  Each of these functions computes the Euclidean distances between two vectors.

The following equation is used:

\[ z[0] = \sqrt{\sum_{i=0}^{n-1} (x[i] - y[i])^2} \]

Parameters  Each of the functions takes the following arguments:

- \( z \)  Pointer to the distance between the two vectors.
- \( x \)  Pointer to the first element of the first source vector.
- \( y \)  Pointer to the first element of the second source vector.
- \( n \)  Number of elements in the vectors.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
Name  
mlib_VectorDotProd_U8_Sat, mlib_VectorDotProd_U8C_Sat,
mlib_VectorDotProd_S8_Sat, mlib_VectorDotProd_S8C_Sat,
mlib_VectorDotProd_S16_Sat, mlib_VectorDotProd_S16C_Sat,
mlib_VectorDotProd_S32_Sat, mlib_VectorDotProd_S32C_Sat – vector dot product (inner
product)

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_VectorDotProd_U8_Sat(mlib_d64 *z,
    const mlib_u8  *x, const mlib_u8  *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_U8C_Sat(mlib_d64 *z,
    const mlib_u8  *x, const mlib_u8  *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_S8_Sat(mlib_d64 *z,
    const mlib_s8  *x, const mlib_s8  *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_S8C_Sat(mlib_d64 *z,
    const mlib_s8  *x, const mlib_s8  *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_S16_Sat(mlib_d64 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_S16C_Sat(mlib_d64 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_S32_Sat(mlib_d64 *z,
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorDotProd_S32C_Sat(mlib_d64 *z,
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

Description  
Each of these functions computes the dot product of two vectors, defined by the following
equation:
\[ Z = X \cdot Y^* \]
where \( Y^* \) is the conjugate of the \( Y \) vector.

For real data, the following equation is used:
\[
\begin{align*}
    n-1 \\
    z[0] &= \text{SUM} (x[i] \cdot y[i]) \\
    i &= 0
\end{align*}
\]

For complex data, the following equation is used:
\[
\begin{align*}
    n-1 \\
    z[0] &= \text{SUM} (x[2\cdot i] \cdot y[2\cdot i] + x[2\cdot i + 1] \cdot y[2\cdot i + 1]) \\
    i &= 0 \\
    n-1
\end{align*}
\]


\[
z[1] = \sum_{i=0} \left( x[2i+1]y[2i] - x[2i]y[2i+1] \right)
\]

**Parameters**
Each of the functions takes the following arguments:

- **z** Pointer to the dot product of the two vectors.
- **x** Pointer to the first element of the first source vector.
- **y** Pointer to the first element of the second source vector.
- **n** Number of elements in the vectors.

**Return Values**
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)
mlib_VectorMag_U8C, mlib_VectorMag_S8C, mlib_VectorMag_S16C, mlib_VectorMag_S32C – vector complex magnitude

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMag_U8C(mlib_d64 *m, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorMag_S8C(mlib_d64 *m, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorMag_S16C(mlib_d64 *m, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorMag_S32C(mlib_d64 *m, const mlib_s32 *x, mlib_s32 n);

Each of these functions computes the magnitude vector of the complex input vector.

The following equation is used:
\[ m[i] = (x[2*i]**2 + x[2*i + 1]**2)**0.5 \]
where \( i = 0, 1, \ldots, (n - 1) \).

Each of the functions takes the following arguments:

\[ m \quad \text{Pointer to the destination magnitude vector.} \]
\[ x \quad \text{Pointer to the source vector} \]
\[ n \quad \text{Number of elements in the vector.} \]

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
mlib_VectorMaximumMag_U8C, mlib_VectorMaximumMag_S8C,
mlib_VectorMaximumMag_S16C, mlib_VectorMaximumMag_S32C,
mlib_VectorMaximumMag_F32C, mlib_VectorMaximumMag_D64C – find the first element
with the maximum magnitude in a vector

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_VectorMaximumMag_U8C(mlib_u8 *max,
    const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorMaximumMag_S8C(mlib_s8 *max,
    const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorMaximumMag_S16C(mlib_s16 *max,
    const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorMaximumMag_S32C(mlib_s32 *max,
    const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorMaximumMag_F32C(mlib_f32 *max,
    const mlib_f32 *x, mlib_s32 n);
mlib_status mlib_VectorMaximumMag_D64C(mlib_d64 *max,
    const mlib_d64 *x, mlib_s32 n);

Description  
Each of these functions finds the first element with the maximum magnitude in a complex
vector, then puts the real and imaginary parts of it into max[0] and max[1], respectively.

Parameters  
Each of the functions takes the following arguments:

max        Pointer to the first element with the maximum magnitude.

x          Pointer to the first element of the source vector.

n          Number of elements in the source vector.

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VectorMinimumMag_U8C(3MLIB), mlib_MatrixMaximumMag_U8C(3MLIB),
mlib_MatrixMinimumMag_U8C(3MLIB), attributes(5)
**Name**
mlib_VectorMaximum_U8, mlib_VectorMaximum_S8, mlib_VectorMaximum_S16, mlib_VectorMaximum_S32, mlib_VectorMaximum_F32, mlib_VectorMaximum_D64 – find the maximum value in a vector.

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMaximum_U8(mlib_u8 *max, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorMaximum_S8(mlib_s8 *max, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorMaximum_S16(mlib_s16 *max, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorMaximum_S32(mlib_s32 *max, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorMaximum_F32(mlib_f32 *max, const mlib_f32 *x, mlib_s32 n);
mlib_status mlib_VectorMaximum_D64(mlib_d64 *max, const mlib_d64 *x, mlib_s32 n);

**Description**
Each of these functions finds the maximum value of all elements in a vector.

The following equation is used:

\[ \text{max}[0] = \text{MAX} \{ x[i] \mid i = 0, 1, \ldots, (n - 1) \} \]

**Parameters**
Each of the functions takes the following arguments:

- **max**  
  Pointer to the maximum value.
- **x**  
  Pointer to the first element of the source vector.
- **n**  
  Number of elements in the source vector.

**Return Values**
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_VectorMinimum_U8(3MLIB), mlib_MatrixMaximum_U8(3MLIB), mlib_MatrixMinimum_U8(3MLIB), attributes(5)
Each of these functions computes the complex vector from two vectors representing the real and imaginary parts. The following equation is used:

\[ z[2k] = r[k] \]
\[ z[2k + 1] = i[k] \]

where \( k = 0, 1, \ldots, (n - 1) \).

### Parameters
Each of the functions takes the following arguments:

- **z** Pointer to the first complex element of the destination vector. \( z[2k] \) contains the real part, and \( z[2k + 1] \) contains the imaginary part.
- **r** Pointer to the first element of the real part.
- **i** Pointer to the first element of the imaginary part.
- **n** Number of elements in the vectors.

### Return Values
Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

### Attributes
See [attributes(5)](attributes(5)) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VectorSplit_U8_U8C(3MLIB), attributes(5)
Name  
mlib_VectorMinimumMag_U8C, mlib_VectorMinimumMag_S8C,  
mlib_VectorMinimumMag_S16C, mlib_VectorMinimumMag_S32C,  
mlib_VectorMinimumMag_F32C, mlib_VectorMinimumMag_D64C – find the first element  
with the minimum magnitude in a vector

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_VectorMinimumMag_U8C(mlib_u8 * min, const mlib_u8 *x,  
mlib_s32 n);
mlib_status mlib_VectorMinimumMag_S8C(mlib_s8 * min, const mlib_s8 *x,  
mlib_s32 n);
mlib_status mlib_VectorMinimumMag_S16C(mlib_s16 *min, const mlib_s16 *x,  
mlib_s32 n);
mlib_status mlib_VectorMinimumMag_S32C(mlib_s32 *min, const mlib_s32 *x,  
mlib_s32 n);
mlib_status mlib_VectorMinimumMag_F32C(mlib_f32 *min, const mlib_f32 *x,  
mlib_s32 n);
mlib_status mlib_VectorMinimumMag_D64C(mlib_d64 *min, const mlib_d64 *x,  
mlib_s32 n);

Description  
Each of these functions finds the first element with the minimum magnitude in a complex  
vector, then puts the real and imaginary parts of it into min[0] and min[1], respectively.

Parameters  
Each of the functions takes the following arguments:  
\textit{min} Pointer to the first element with the minimum magnitude.  
\textit{x} Pointer to the first element of the source vector.  
\textit{n} Number of elements in the source vector.

Return Values  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VectorMaximumMag_U8C(3MLIB), mlib_MatrixMaximumMag_U8C(3MLIB),  
mlib_MatrixMinimumMag_U8C(3MLIB), attributes(5)
mlib_VectorMinimum_U8, mlib_VectorMinimum_S8, mlib_VectorMinimum_S16, mlib_VectorMinimum_S32, mlib_VectorMinimum_F32, mlib_VectorMinimum_D64 – find the minimum value in a vector

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMinimum_U8(mlib_u8 *min, const mlib_u8 *x, mlib_s32 n);
mlib_status mlib_VectorMinimum_S8(mlib_s8 *min, const mlib_s8 *x, mlib_s32 n);
mlib_status mlib_VectorMinimum_S16(mlib_s16 *min, const mlib_s16 *x, mlib_s32 n);
mlib_status mlib_VectorMinimum_S32(mlib_s32 *min, const mlib_s32 *x, mlib_s32 n);
mlib_status mlib_VectorMinimum_F32(mlib_f32 *min, const mlib_f32 *x, mlib_s32 n);
mlib_status mlib_VectorMinimum_D64(mlib_d64 *min, const mlib_d64 *x, mlib_s32 n);

Description
Each of these functions finds the minimum value of all elements in a vector.

The following equation is used:
max[0] = MIN\{ x[i] \mid i = 0, 1, \ldots, (n - 1) \}

Parameters
Each of the functions takes the following arguments:

min Pointer to the minimum value.
x Pointer to the first element of the source vector.
n Number of elements in the source vector.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VectorMaximum_U8(3MLIB), mlib_MatrixMaximum_U8(3MLIB), mlib_MatrixMinimum_U8(3MLIB), attributes(5)
Name  
mlib_VectorMulMShift_S16_S16_Mod, mlib_VectorMulMShift_S16_S16_Sat, 
mlib_VectorMulMShift_S16C_S16C_Mod, mlib_VectorMulMShift_S16C_S16C_Sat – multiplication of vector by matrix with shifting

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_VectorMulMShift_S16_S16_Mod(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m, 
    mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulMShift_S16_S16_Sat(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m, 
    mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulMShift_S16C_S16C_Mod(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m, 
    mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulMShift_S16C_S16C_Sat(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 m, 
    mlib_s32 n, mlib_s32 shift);

Description
Each of these functions multiplies a vector by a matrix and shifts the results.

For real data, the following equation is used:

\[ z[i] = \sum_{j=0}^{m-1} x[j] \cdot y[j \cdot m + i] \cdot 2^{-shift} \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[ z[2i] = \sum_{j=0}^{m-1} (xR \cdot yR - xI \cdot yI) \cdot 2^{-shift} \]
\[ z[2i + 1] = \sum_{j=0}^{m-1} (xR \cdot yI + xI \cdot yR) \cdot 2^{-shift} \]

where \( i = 0, 1, \ldots, (n - 1) \), and

\( xR = x[2*j] \)
\( xI = x[2*j + 1] \)
\( yR = y[2*(j*m + i)] \)
\( yI = y[2*(j*m + i) + 1] \)
Parameters Each of the functions takes the following arguments:
  
  \( z \) Pointer to the first element of the destination vector.
  
  \( x \) Pointer to the first element of the source vector.
  
  \( y \) Pointer to the first element of the source matrix.
  
  \( m \) Number of rows in the matrix, and number of elements in the source vector.
  
  \( n \) Number of columns in the matrix, and number of elements in the destination vector.
  
  \( \text{shift} \) Right shifting factor.

Return Values Each of the functions returns \texttt{MLIB\_SUCCESS} if successful. Otherwise it returns \texttt{MLIB\_FAILURE}.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also \texttt{mlib\_VectorMulM\_U8\_U8\_Mod(3MLIB)}, attributes(5)
Name  mlib_VectorMulM_U8_U8_Mod, mlib_VectorMulM_U8_U8_Sat,
      mlib_VectorMulM_S8_S8_Mod, mlib_VectorMulM_S8_S8_Sat,
      mlib_VectorMulM_S8C_S8C_Mod, mlib_VectorMulM_S8C_S8C_Sat,
      mlib_VectorMulM_S16_U8_Mod, mlib_VectorMulM_S16_U8_Sat,
      mlib_VectorMulM_S16_S8_Mod, mlib_VectorMulM_S16_S8_Sat,
      mlib_VectorMulM_S16_S16_Mod, mlib_VectorMulM_S16_S16_Sat,
      mlib_VectorMulM_S16C_U8C_Mod, mlib_VectorMulM_S16C_U8C_Sat,
      mlib_VectorMulM_S16C_S8C_Mod, mlib_VectorMulM_S16C_S8C_Sat,
      mlib_VectorMulM_S32_S16_Mod, mlib_VectorMulM_S32_S16_Sat,
      mlib_VectorMulM_S32_S32_Mod, mlib_VectorMulM_S32_S32_Sat

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulM_U8_U8_Mod(mlib_u8 *z,
    const mlib_u8 *x,
    const mlib_u8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_U8_U8_Sat(mlib_u8 *z,
    const mlib_u8 *x,
    const mlib_u8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_U8C_U8C_Mod(mlib_u8 *z,
    const mlib_u8 *x,
    const mlib_u8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_U8C_U8C_Sat(mlib_u8 *z,
    const mlib_u8 *x,
    const mlib_u8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S8_S8_Mod(mlib_s8 *z,
    const mlib_s8 *x,
    const mlib_s8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S8_S8_Sat(mlib_s8 *z,
    const mlib_s8 *x,
    const mlib_s8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S8C_S8C_Mod(mlib_s8 *z,
    const mlib_s8 *x,
    const mlib_s8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S8C_S8C_Sat(mlib_s8 *z,
    const mlib_s8 *x,
    const mlib_s8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S16_U8_Mod(mlib_s16 *z,
    const mlib_u8 *x,
    const mlib_u8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S16_U8_Sat(mlib_s16 *z,
    const mlib_u8 *x,
    const mlib_u8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S16_S8_Mod(mlib_s16 *z,
    const mlib_s8 *x,
    const mlib_s8 *y,
    mlib_s32 m,
    mlib_s32 n);

mlib_status mlib_VectorMulM_S16_S8_Sat(mlib_s16 *z,
    const mlib_s8 *x,
    const mlib_s8 *y,
    mlib_s32 m,
    mlib_s32 n);
Description  Each of these functions multiplies a vector by a matrix.

For real data, the following equation is used:
\[ z[i] = \sum_{j=0}^{m-1} (x[j] \times y[j \times m + i]) \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[ z[2i] = \sum_{j=0}^{m-1} (xR \times yR - xI \times yI) \]
\[ z[2i + 1] = \sum_{j=0}^{m-1} (xR \times yI + xI \times yR) \]

where \( i = 0, 1, \ldots, (n - 1) \), and

- \( xR = x[2j] \)
- \( xI = x[2j + 1] \)
- \( yR = y[2(j \times m + i)] \)
- \( yI = y[2(j \times m + i) + 1] \)

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the destination vector.
- \( x \) Pointer to the first element of the source vector.
- \( y \) Pointer to the first element of the source matrix.
- \( m \) Number of rows in the matrix, and number of elements in the source vector.
- \( n \) Number of columns in the matrix, and number of elements in the destination vector.

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_VectorMulMShift_S16_S16_Mod(3MLIB), attributes(5)
mlib_VectorMulSAdd_U8_Mod, mlib_VectorMulSAdd_U8_Sat,
mlib_VectorMulSAdd_U8C_Mod, mlib_VectorMulSAdd_U8C_Sat,
mlib_VectorMulSAdd_S8_Mod, mlib_VectorMulSAdd_S8_Sat,
mlib_VectorMulSAdd_S8C_Mod, mlib_VectorMulSAdd_S8C_Sat,
mlib_VectorMulSAdd_S16_Mod, mlib_VectorMulSAdd_S16_Sat,
mlib_VectorMulSAdd_S16C_Mod, mlib_VectorMulSAdd_S16C_Sat,
mlib_VectorMulSAdd_S32_Mod, mlib_VectorMulSAdd_S32_Sat,
mlib_VectorMulSAdd_S32C_Mod, mlib_VectorMulSAdd_S32C_Sat – vector multiplication
by scalar plus addition, in place

Synopsis

c [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulSAdd_U8_Mod(mlib_u8 *xz,
const mlib_u8 *y, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_U8_Sat(mlib_u8 *xz,
const mlib_u8 *y, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_U8C_Mod(mlib_u8 *xz,
const mlib_u8 *y, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_U8C_Sat(mlib_u8 *xz,
const mlib_u8 *y, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S8_Mod(mlib_s8 *xz,
const mlib_s8 *y, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S8_Sat(mlib_s8 *xz,
const mlib_s8 *y, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S8C_Mod(mlib_s8 *xz,
const mlib_s8 *y, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S8C_Sat(mlib_s8 *xz,
const mlib_s8 *y, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S16_Mod(mlib_s16 *xz,
const mlib_s16 *y, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S16_Sat(mlib_s16 *xz,
const mlib_s16 *y, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S16C_Mod(mlib_s16 *xz,
const mlib_s16 *y, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S16C_Sat(mlib_s16 *xz,
const mlib_s16 *y, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S32_Mod(mlib_s32 *xz,
const mlib_s32 *y, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S32_Sat(mlib_s32 *xz,
const mlib_s32 *y, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S32C_Mod(mlib_s32 *xz, const mlib_s32 *y, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S32C_Sat(mlib_s32 *xz, const mlib_s32 *y, const mlib_s32 *c, mlib_s32 n);

**Description**  Each of these functions computes an in-place multiplication of a vector by a scalar and adds the result to another vector.

For real data, the following equation is used:

\[ xz[i] = xz[i] + y[i]*c[0] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**  Each of the functions takes the following arguments:

- `xz`  Pointer to the first element of the first source and destination vector.
- `y`  Pointer to the first element of the second source vector.
- `c`  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the real part of the scalar, and \( c[1] \) contains the imaginary part of the scalar.
- `n`  Number of elements in the vectors.

**Return Values**  Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  `mlib_VectorMulSAdd_U8_U8_Mod(3MLIB), attributes(5)`
mlib_VectorMulSAdd_U8_U8_Mod, mlib_VectorMulSAdd_U8_U8_Sat, mlib_VectorMulSAdd_S8_S8_Mod, mlib_VectorMulSAdd_S8_S8_Sat, mlib_VectorMulSAdd_S16_S16_Mod, mlib_VectorMulSAdd_S16_S16_Sat, mlib_VectorMulSAdd_S32_S32_Mod, mlib_VectorMulSAdd_S32_S32_Sat – vector multiplication by scalar plus addition

### Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulSAdd_U8_U8_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_U8_U8_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S8_S8_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S8_S8_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_S16_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_S16_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32_S32_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, const mlib_s32 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32_S32_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, const mlib_s32 *c,
    mlib_s32 n);
```

### Notes
mlib_status mlib_VectorMulSAdd_S16_U8_Mod(mlib_s16 *z, 
    const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_U8_Sat(mlib_s16 *z, 
    const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_S8_Mod(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_S8_Sat(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_S16_Mod(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16_S16_Sat(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16C_U8C_Mod(mlib_s16 *z, 
    const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16C_U8C_Sat(mlib_s16 *z, 
    const mlib_u8 *x, const mlib_u8 *y, const mlib_u8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16C_S8C_Mod(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16C_S8C_Sat(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, const mlib_s8 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16C_S16C_Mod(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S16C_S16C_Sat(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32_S16_Mod(mlib_s32 *z, 
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c, 
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32_S16_Sat(mlib_s32 *z, 
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c, 
    mlib_s32 n);
mlib_status mlib_VectorMulSAdd_S32_S32_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, const mlib_s32 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32_S32_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, const mlib_s32 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32C_S16C_Mod(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32C_S16C_Sat(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *y, const mlib_s16 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32C_S32C_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, const mlib_s32 *c,
    mlib_s32 n);

mlib_status mlib_VectorMulSAdd_S32C_S32C_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, const mlib_s32 *c,
    mlib_s32 n);

Description  Each of these functions multiplies a vector by a scalar and adds the result to another vector.

For real data, the following equation is used:

\[ z[i] = x[i] + y[i] \times c[0] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
    z[2i + 1] &= x[2i + 1] + y[2i] \times c[1] + y[2i + 1] \times c[0]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

Parameters  Each of the functions takes the following arguments:

- \( z \)  Pointer to the first element of the destination vector.
- \( x \)  Pointer to the first element of the first source vector.
- \( y \)  Pointer to the first element of the second source vector.
- \( c \)  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the real part of the scalar, and \( c[1] \) contains the imaginary part of the scalar.
- \( n \)  Number of elements in the vectors.
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_VectorMulSAdd_U8_U8_Mod(3MLIB), attributes(5)`
Name  mllib_VectorMulShift_U8_Mod, mllib_VectorMulShift_U8_Sat,  
      mllib_VectorMulShift_U8C_Mod, mllib_VectorMulShift_U8C_Sat,  
      mllib_VectorMulShift_S8_Mod, mllib_VectorMulShift_S8_Sat,  
      mllib_VectorMulShift_S8C_Mod, mllib_VectorMulShift_S8C_Sat,  
      mllib_VectorMulShift_S16_Mod, mllib_VectorMulShift_S16_Sat,  
      mllib_VectorMulShift_S16C_Mod, mllib_VectorMulShift_S16C_Sat,  
      mllib_VectorMulShift_S32_Mod, mllib_VectorMulShift_S32_Sat,  
      mllib_VectorMulShift_S32C_Mod, mllib_VectorMulShift_S32C_Sat – vector multiplication  
with shifting, in place

Synopsis  cc [-f... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mllib_VectorMulShift_U8_Mod(mlib_u8 *xz,  
const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_U8_Sat(mlib_u8 *xz,  
const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_U8C_Mod(mlib_u8 *xz,  
const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_U8C_Sat(mlib_u8 *xz,  
const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S8_Mod(mlib_s8 *xz,  
const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S8_Sat(mlib_s8 *xz,  
const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S8C_Mod(mlib_s8 *xz,  
const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S8C_Sat(mlib_s8 *xz,  
const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S16_Mod(mlib_s16 *xz,  
const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S16_Sat(mlib_s16 *xz,  
const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S16C_Mod(mlib_s16 *xz,  
const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S16C_Sat(mlib_s16 *xz,  
const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S32_Mod(mlib_s32 *xz,  
const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mllib_VectorMulShift_S32_Sat(mlib_s32 *xz,  
const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S32C_Mod(mlib_s32 *xz,  
   const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulShift_S32C_Sat(mlib_s32 *xz,  
   const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);

**Description**  
Each of these functions performs an in-place multiplication of two vectors and shifts the result.

For real data, the following equation is used:

\[ xz[i] = xz[i] \cdot y[i] \cdot 2^{-\text{shift}} \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
tmp & = xz[2*i] \\
xz[2*i] & = (tmp \cdot y[2*i] - xz[2*i + 1] \cdot y[2*i + 1]) \cdot 2^{-\text{shift}} \\
xz[2*i + 1] & = (tmp \cdot y[2*i + 1] + xz[2*i + 1] \cdot y[2*i]) \cdot 2^{-\text{shift}}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**  
Each of the functions takes the following arguments:

- \( xz \)  
  Pointer to the first element of the first source and result vector.
- \( y \)  
  Pointer to the first element of the second source vector.
- \( n \)  
  Number of elements in each vector.
- \( shift \)  
  Right shifting factor. The ranges of valid shift are:
  
  \[
  1 \leq \text{shift} \leq \begin{cases} 
  8 & \text{for U8, S8, U8C, S8C types} \\
  16 & \text{for S16, S16C types} \\
  31 & \text{for S32, S32C types}
  \end{cases}
  \]

**Return Values**  
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VectorMulShift_U8_U8_Mod(3MLIB), attributes(5)
mlib_VectorMulShift_U8_U8_Mod, mlib_VectorMulShift_U8_U8_Sat,
mlib_VectorMulShift_U8C_U8C_Mod, mlib_VectorMulShift_U8C_U8C_Sat,
mlib_VectorMulShift_S8_S8_Mod, mlib_VectorMulShift_S8_S8_Sat,
mlib_VectorMulShift_S8C_S8C_Mod, mlib_VectorMulShift_S8C_S8C_Sat,
mlib_VectorMulShift_S16_S16_Mod, mlib_VectorMulShift_S16_S16_Sat,
mlib_VectorMulShift_S16C_S16C_Mod, mlib_VectorMulShift_S16C_S16C_Sat,
mlib_VectorMulShift_S32_S32_Mod, mlib_VectorMulShift_S32_S32_Sat,
mlib_VectorMulShift_S32C_S32C_Mod, mlib_VectorMulShift_S32C_S32C_Sat – vector
multiplication with shifting

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulShift_U8_U8_Mod(mlib_u8 *z,
                                             const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_U8_U8_Sat(mlib_u8 *z,
                                             const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_U8C_U8C_Mod(mlib_u8 *z,
                                             const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_U8C_U8C_Sat(mlib_u8 *z,
                                             const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S8_S8_Mod(mlib_s8 *z,
                                             const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S8_S8_Sat(mlib_s8 *z,
                                             const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S8C_S8C_Mod(mlib_s8 *z,
                                             const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S8C_S8C_Sat(mlib_s8 *z,
                                             const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S16_S16_Mod(mlib_s16 *z,
                                             const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S16_S16_Sat(mlib_s16 *z,
                                             const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S16C_S16C_Mod(mlib_s16 *z,
                                             const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S16C_S16C_Sat(mlib_s16 *z,
                                             const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S32_S32_Mod(mlib_s32 *z,
                                             const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S32_S32_Sat(mlib_s32 *z,
                                             const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulShift_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n, mlib_s32 shift);

Description Each of these functions performs a multiplication of two vectors, shifts the result, and puts it into a third vector.

For real data, the following equation is used:

\[ z[i] = x[i] * y[i] * 2^{(-shift)} \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

Parameters Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the result vector.
- \( x \) Pointer to the first element of the first source vector.
- \( y \) Pointer to the first element of the second source vector.
- \( n \) Number of elements in each vector.
- \( shift \) Right shifting factor. The ranges of valid \( shift \) are:
  
  - \( 1 \leq shift \leq 8 \) for U8, S8, U8C, S8C types
  - \( 1 \leq shift \leq 16 \) for S16, S16C types
  - \( 1 \leq shift \leq 31 \) for S32, S32C types

Return Values Each of the functions returns \textbf{MLIB_SUCCESS} if successful. Otherwise it returns \textbf{MLIB_FAILURE}.

Attributes See attributes\textit{(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also \textit{mlib_VectorMulShift_U8_Mod(3MLIB)}, attributes\textit{(5)}
mlib_VectorMulSShift_U8_Mod, mlib_VectorMulSShift_U8_Sat, mlib_VectorMulSShift_U8C_Mod, mlib_VectorMulSShift_U8C_Sat, mlib_VectorMulSShift_S8_Mod, mlib_VectorMulSShift_S8_Sat, mlib_VectorMulSShift_S8C_Mod, mlib_VectorMulSShift_S8C_Sat, mlib_VectorMulSShift_S16_Mod, mlib_VectorMulSShift_S16_Sat, mlib_VectorMulSShift_S16C_Mod, mlib_VectorMulSShift_S16C_Sat, mlib_VectorMulSShift_S32_Mod, mlib_VectorMulSShift_S32_Sat, mlib_VectorMulSShift_S32C_Mod, mlib_VectorMulSShift_S32C_Sat – vector multiplication by scalar plus shifting, in place

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulSShift_U8_Mod(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_U8_Sat(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_U8C_Mod(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_U8C_Sat(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8_Mod(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8_Sat(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8C_Mod(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8C_Sat(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16_Mod(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16_Sat(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16C_Mod(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16C_Sat(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32_Mod(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32_Sat(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32C_Mod(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32C_Sat(mlib_s32 *xz, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

**Description**

Each of these functions performs an in-place multiplication of a vector by a scalar and shifts the result.

For real data, the following equation is used:

\[ xz[i] = xz[i] \times c[0] \times 2^{-shift} \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
tmp &= xz[2i] \\
xz[2i] &= (tmp \times c[0] - xz[2i + 1] \times c[1]) \times 2^{-shift} \\
xz[2i + 1] &= (tmp \times c[1] + xz[2i + 1] \times c[0]) \times 2^{-shift}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**

Each of the functions takes the following arguments:

- **xz**: Pointer to the first element of the source and result vector.
- **c**: Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the real part of the scalar, and \( c[1] \) contains the imaginary part of the scalar.
- **n**: Number of elements in each vector.
- **shift**: Right shifting factor. The ranges of valid shift are:
  
  - \( 1 \leq \text{shift} \leq 8 \) for U8, S8, U8C, S8C types
  - \( 1 \leq \text{shift} \leq 16 \) for S16, S16C types
  - \( 1 \leq \text{shift} \leq 31 \) for S32, S32C types

**Return Values**

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_VectorMulSShift_U8_U8_Mod(3MLIB), attributes(5)
**Name**
mlib_VectorMulSShift_U8_U8_Mod, mlib_VectorMulSShift_U8_U8_Sat,
mlib_VectorMulSShift_U8C_U8C_Mod, mlib_VectorMulSShift_U8C_U8C_Sat,
mlib_VectorMulSShift_S8_S8_Mod, mlib_VectorMulSShift_S8_S8_Sat,
mlib_VectorMulSShift_S8C_S8C_Mod, mlib_VectorMulSShift_S8C_S8C_Sat,
mlib_VectorMulSShift_S16_S16_Mod, mlib_VectorMulSShift_S16_S16_Sat,
mlib_VectorMulSShift_S16C_S16C_Mod, mlib_VectorMulSShift_S16C_S16C_Sat,
mlib_VectorMulSShift_S32_S32_Mod, mlib_VectorMulSShift_S32_S32_Sat,
mlib_VectorMulSShift_S32C_S32C_Mod, mlib_VectorMulSShift_S32C_S32C_Sat – vector
multiplication by scalar plus shifting

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulSShift_U8_U8_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_U8_U8_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_U8C_U8C_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_U8C_U8C_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8_S8_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8_S8_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8C_S8C_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S8C_S8C_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16_S16_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16_S16_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16C_S16C_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S16C_S16C_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32_S32_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32_S32_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);
mlib_status mlib_VectorMulSShift_S32C_S32C_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

mlib_status mlib_VectorMulSShift_S32C_S32C_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n, mlib_s32 shift);

**Description**  Each of these functions performs a multiplication of a vector by a scalar and shifts the result.

For real data, the following equation is used:

\[ z[i] = x[i] \times c[0] \times 2^{-\text{shift}} \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
    z[2i] &= (x[2i]\times c[0] - x[2i + 1]\times c[1]) \times 2^{-\text{shift}} \\
    z[2i + 1] &= (x[2i]\times c[1] + x[2i + 1]\times c[0]) \times 2^{-\text{shift}}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**  Each of the functions takes the following arguments:

- \( z \)  Pointer to the first element of the result vector.
- \( x \)  Pointer to the first element of the source vector.
- \( c \)  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the real part of the scalar, and \( c[1] \) contains the imaginary part of the scalar.
- \( n \)  Number of elements in each vector.
- \( \text{shift} \)  Right shifting factor. The ranges of valid \( \text{shift} \) are:

  - \( 1 \leq \text{shift} \leq 8 \)  for U8, S8, U8C, S8C types
  - \( 1 \leq \text{shift} \leq 16 \)  for S16, S16C types
  - \( 1 \leq \text{shift} \leq 31 \)  for S32, S32C types

**Return Values**  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_VectorMulSShift_U8_U8_Mod(3MLIB), attributes(5)
mlib_VectorMulS_U8_Mod, mlib_VectorMulS_U8_Sat, mlib_VectorMulS_U8C_Mod, mlib_VectorMulS_U8C_Sat, mlib_VectorMulS_S8_Mod, mlib_VectorMulS_S8_Sat, mlib_VectorMulS_S8C_Mod, mlib_VectorMulS_S8C_Sat, mlib_VectorMulS_S16_Mod, mlib_VectorMulS_S16_Sat, mlib_VectorMulS_S16C_Mod, mlib_VectorMulS_S16C_Sat, mlib_VectorMulS_S32_Mod, mlib_VectorMulS_S32_Sat, mlib_VectorMulS_S32C_Mod, mlib_VectorMulS_S32C_Sat – vector multiplication by scalar, in place

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulS_U8_Mod(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_U8_Sat(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_U8C_Mod(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_U8C_Sat(mlib_u8 *xz,
    const mlib_u8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S8_Mod(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S8_Sat(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S8C_Mod(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S8C_Sat(mlib_s8 *xz,
    const mlib_s8 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16_Mod(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16_Sat(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_Mod(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S16C_Sat(mlib_s16 *xz,
    const mlib_s16 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32_Mod(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32_Sat(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_Mod(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n);

mlib_status mlib_VectorMulS_S32C_Sat – vector multiplication by scalar, in place

#include <mlib.h>
mlib_status mlib_VectorMulS_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *c, mlib_s32 n);

**Description**  Each of these functions computes an in-place multiplication of a vector by a scalar.

For real data, the following equation is used:

\[ xz[i] = xz[i] \times c[0] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
    \text{tmp} & = xz[2*i] \\
    xz[2*i] & = \text{tmp} \times c[0] - xz[2*i + 1] \times c[1] \\
    xz[2*i + 1] & = \text{tmp} \times c[1] + xz[2*i + 1] \times c[0]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**  Each of the functions takes the following arguments:

- **xz**  Pointer to the first element of the source and destination vector.
- **c**  Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the real part of the scalar, and \( c[1] \) contains the imaginary part of the scalar.
- **n**  Number of elements in the vectors.

**Return Values**  Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_VectorMulS_U8_U8_Mod(3MLIB), attributes(5)
Name  
mlib_VectorMulS_U8_U8_Mod, mlib_VectorMulS_U8_U8_Sat,
mlib_VectorMulS_U8C_U8C_Mod, mlib_VectorMulS_U8C_U8C_Sat,
mlib_VectorMulS_S8_S8_Mod, mlib_VectorMulS_S8_S8_Sat,
mlib_VectorMulS_S8C_S8C_Mod, mlib_VectorMulS_S8C_S8C_Sat,
mlib_VectorMulS_S16_U8_Mod, mlib_VectorMulS_S16_U8_Sat,
mlib_VectorMulS_S16_S8_Mod, mlib_VectorMulS_S16_S8_Sat,
mlib_VectorMulS_S16_S16_Mod, mlib_VectorMulS_S16_S16_Sat,
mlib_VectorMulS_S16C_U8C_Mod, mlib_VectorMulS_S16C_U8C_Sat,
mlib_VectorMulS_S16C_S8C_Mod, mlib_VectorMulS_S16C_S8C_Sat,
mlib_VectorMulS_S16C_S16C_Mod, mlib_VectorMulS_S16C_S16C_Sat,
mlib_VectorMulS_S32_S16_Mod, mlib_VectorMulS_S32_S16_Sat,
mlib_VectorMulS_S32_S32_Mod, mlib_VectorMulS_S32_S32_Sat,
mlib_VectorMulS_S32C_S16C_Mod, mlib_VectorMulS_S32C_S16C_Sat,
mlib_VectorMulS_S32C_S32C_Mod, mlib_VectorMulS_S32C_S32C_Sat – vector
multiplication by scalar

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMulS_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_U8_Mod(mlib_s16 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_U8_Sat(mlib_s16 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_S8_Mod(mlib_s16 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x,
const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x,
const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x,
const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x,
const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x,
const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorMulS_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);

**Description** Each of these functions multiplies a vector by a scalar.

For real data, the following equation is used:

\[ z[i] = x[i] \times c[0] \]
where $i = 0, 1, \ldots, (n - 1)$.

For complex data, the following equation is used:

\[
\begin{align*}
z[2i] &= x[2i] \cdot c[0] - x[2i + 1] \cdot c[1] \\
z[2i + 1] &= x[2i] \cdot c[1] + x[2i + 1] \cdot c[0]
\end{align*}
\]

where $i = 0, 1, \ldots, (n - 1)$.

**Parameters**

Each of the functions takes the following arguments:

- $z$ Pointer to the first element of the destination vector.
- $x$ Pointer to the first element of the source vector.
- $c$ Pointer to the source scalar. When the function is used with complex data types, $c[0]$ contains the real part of the scalar, and $c[1]$ contains the imaginary part of the scalar.
- $n$ Number of elements in the vectors.

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`mlib_VectorMulS_U8_U8_Mod(3MLIB)`, attributes(5)
mlib_VectorMul_U8_Mod, mlib_VectorMul_U8_Sat, mlib_VectorMul_U8C_Mod,
mlib_VectorMul_U8C_Sat, mlib_VectorMul_S8_Mod, mlib_VectorMul_S8_Sat,
mlib_VectorMul_S8C_Mod, mlib_VectorMul_S8C_Sat, mlib_VectorMul_S16_Mod,
mlib_VectorMul_S16_Sat, mlib_VectorMul_S16C_Mod, mlib_VectorMul_S16C_Sat,
mlib_VectorMul_S32_Mod, mlib_VectorMul_S32_Sat, mlib_VectorMul_S32C_Mod,
mlib_VectorMul_S32C_Sat – vector multiplication, in place

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorMul_U8_Mod(mlib_u8 *xz, const mlib_u8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_U8_Sat(mlib_u8 *xz, const mlib_u8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_U8C_Mod(mlib_u8 *xz, const mlib_u8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_U8C_Sat(mlib_u8 *xz, const mlib_u8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S8_Mod(mlib_s8 *xz, const mlib_s8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S8_Sat(mlib_s8 *xz, const mlib_s8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S8C_Mod(mlib_s8 *xz, const mlib_s8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S8C_Sat(mlib_s8 *xz, const mlib_s8 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S16_Mod(mlib_s16 *xz, const mlib_s16 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S16_Sat(mlib_s16 *xz, const mlib_s16 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S16C_Mod(mlib_s16 *xz, const mlib_s16 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S16C_Sat(mlib_s16 *xz, const mlib_s16 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S32_Mod(mlib_s32 *xz, const mlib_s32 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S32_Sat(mlib_s32 *xz, const mlib_s32 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S32C_Mod(mlib_s32 *xz, const mlib_s32 *y,
        mlib_s32 n);
mlib_status mlib_VectorMul_S32C_Sat(mlib_s32 *xz, const mlib_s32 *y, mlib_s32 n);

**Description**  Each of these functions performs an in-place multiplication of one vector by another vector.

For real data, the following equation is used:

\[ xz[i] = xz[i] \times y[i] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
\text{tmp} & = xz[2*i] \\
xz[2*i + 1] & = \text{tmp} \times y[2*i + 1] + xz[2*i + 1] \times y[2*i]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**  Each of the functions takes the following arguments:

- \( xz \)  Pointer to the first element of the first source and destination vector.
- \( y \)  Pointer to the first element of the second source vector.
- \( n \)  Number of elements in the vectors.

**Return Values**  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  mlib_VectorMul_U8_U8_Mod(3MLIB), attributes(5)
**Name**  
mlib_VectorMul_U8_U8_Mod, mlib_VectorMul_U8_U8_Sat,  
mlib_VectorMul_U8C_U8C_Mod, mlib_VectorMul_U8C_U8C_Sat,  
mlib_VectorMul_S8_S8_Mod, mlib_VectorMul_S8_S8_Sat,  
mlib_VectorMul_S8C_S8C_Mod, mlib_VectorMul_S8C_S8C_Sat,  
mlib_VectorMul_S16_U8_Mod, mlib_VectorMul_S16_U8_Sat,  
mlib_VectorMul_S16_S8_Mod, mlib_VectorMul_S16_S8_Sat,  
mlib_VectorMul_S16_S16_Mod, mlib_VectorMul_S16_S16_Sat,  
mlib_VectorMul_S16C_U8C_Mod, mlib_VectorMul_S16C_U8C_Sat,  
mlib_VectorMul_S16C_S8C_Mod, mlib_VectorMul_S16C_S8C_Sat,  
mlib_VectorMul_S16C_S16C_Mod, mlib_VectorMul_S16C_S16C_Sat,  
mlib_VectorMul_S32_S16_Mod, mlib_VectorMul_S32_S16_Sat,  
mlib_VectorMul_S32_S32_Mod, mlib_VectorMul_S32_S32_Sat,  
mlib_VectorMul_S32C_S16C_Mod, mlib_VectorMul_S32C_S16C_Sat,  
mlib_VectorMul_S32C_S32C_Mod, mlib_VectorMul_S32C_S32C_Sat – vector multiplication

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_VectorMul_U8_U8_Mod(mlib_u8 *z,  
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_U8_U8_Sat(mlib_u8 *z,  
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_U8C_U8C_Mod(mlib_u8 *z,  
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_U8C_U8C_Sat(mlib_u8 *z,  
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S8_S8_Mod(mlib_s8 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S8_S8_Sat(mlib_s8 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S8C_S8C_Mod(mlib_s8 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S8C_S8C_Sat(mlib_s8 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S16_S8_Mod(mlib_s16 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S16_S8_Sat(mlib_s16 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S16C_S16C_Mod(mlib_s16 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S16C_S16C_Sat(mlib_s16 *z,  
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S32_S16_Mod(mlib_s32 *z,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorMul_S32_S16_Sat(mlib_s32 *z,  
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
```

Multimedia Library Functions - Part 6 1211
mlib_status mlib_VectorMul_S16_S8_Sat(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16_S16_Mod(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16_S16_Sat(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16C_U8C_Mod(mlib_s16 *z, 
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16C_U8C_Sat(mlib_s16 *z, 
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16C_S8C_Mod(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16C_S8C_Sat(mlib_s16 *z, 
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16C_S16C_Mod(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S16C_S16C_Sat(mlib_s16 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32_S16_Mod(mlib_s32 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32_S16_Sat(mlib_s32 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32_S32_Mod(mlib_s32 *z, 
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32_S32_Sat(mlib_s32 *z, 
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32C_S16C_Mod(mlib_s32 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32C_S16C_Sat(mlib_s32 *z, 
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32C_S32C_Mod(mlib_s32 *z, 
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

mlib_status mlib_VectorMul_S32C_S32C_Sat(mlib_s32 *z, 
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

Each of these functions multiplies one vector by another vector.

For real data, the following equation is used:

\[ z[i] = x[i] \times y[i] \]
where $i = 0, 1, \ldots, (n - 1)$.

For complex data, the following equation is used:

\[
\begin{align*}
\end{align*}
\]

where $i = 0, 1, \ldots, (n - 1)$.

**Parameters** Each of the functions takes the following arguments:

- $z$ Pointer to the first element of the destination vector.
- $x$ Pointer to the first element of the first source vector.
- $y$ Pointer to the first element of the second source vector.
- $n$ Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_VectorMul_U8_U8_Mod(3MLIB), attributes(5)
Name  
mlib_VectorNorm_U8_Sat, mlib_VectorNorm_S8_Sat, mlib_VectorNorm_S16_Sat,
mlib_VectorNorm_S32_Sat – vector norm

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorNorm_U8_Sat(mlib_d64 *z, const mlib_u8 *x,
        mlib_s32 n);
mlib_status mlib_VectorNorm_S8_Sat(mlib_d64 *z, const mlib_s8 *x,
        mlib_s32 n);
mlib_status mlib_VectorNorm_S16_Sat(mlib_d64 *z, const mlib_s16 *x,
        mlib_s32 n);
mlib_status mlib_VectorNorm_S32_Sat(mlib_d64 *z, const mlib_s32 *x,
        mlib_s32 n);

Description  
Each of these functions computes the vector normal.

The following equation is used:

\[ z[0] = \left( \sum_{i=0}^{n-1} x[i]^2 \right)^{0.5} \]

Parameters  
Each of the functions takes the following arguments:

- \( z \)  
  Pointer to the norm of the vector.

- \( x \)  
  Pointer to the first element of the source vector.

- \( n \)  
  Number of elements in the vectors.

Return Values  
Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

Attributes  
See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
\texttt{attributes(5)}
mlib_VectorReverseByteOrder() function changes the encoding of each element from big endian to little endian, or from little endian to big endian.

It copies and reverses the byte order of each element of the input vector into the output vector.

**Parameters**

The function takes the following arguments:
- `z`  Pointer to the output vector.
- `x`  Pointer to the input vector.
- `n`  Number of elements in the vectors.
- `s`  Size of elements in bytes.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_VectorReverseByteOrder_Inp(3MLIB),
mlib_VectorReverseByteOrder_S16(3MLIB),
mlib_VectorReverseByteOrder_S16_S16(3MLIB), attributes(5)
mlib_VectorReverseByteOrder_Inp(3MLIB)

Name  mlib_VectorReverseByteOrder_Inp – reverse byte order of vector, in place

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>

          mlib_status mlib_VectorReverseByteOrder_Inp(void *xz,
              mlib_s32 n, mlib_s32 s);

Description  The mlib_VectorReverseByteOrder_Inp() function changes the encoding of each element
              from big endian to little endian, or from little endian to big endian.

              It reverses the byte order of each element of the vector, in place.

Parameters  The function takes the following arguments:
              xz    Pointer to the input and output vector.
              n     Number of elements in the vectors.
              s     Size of elements in bytes.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VectorReverseByteOrder(3MLIB), mlib_VectorReverseByteOrder_S16(3MLIB),
          mlib_VectorReverseByteOrder_S16_S16(3MLIB), attributes(5)
Name mlib_VectorReverseByteOrder_S16, mlib_VectorReverseByteOrder_U16, 
mlib_VectorReverseByteOrder_S32, mlib_VectorReverseByteOrder_U32, 
mlib_VectorReverseByteOrder_S64, mlib_VectorReverseByteOrder_U64, 
mlib_VectorReverseByteOrder_F32, mlib_VectorReverseByteOrder_D64 – reverse byte 
order of vector, in place

Synopsis cc [ flag... ] file... -lmlib [ library... ] 
#include <mlib.h>

mlib_status mlib_VectorReverseByteOrder_S16(mlib_s16 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_U16(mlib_u16 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_S32(mlib_s32 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_U32(mlib_u32 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_S64(mlib_s64 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_U64(mlib_u64 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_F32(mlib_f32 *xz, 
mlib_s32 n);
mlib_status mlib_VectorReverseByteOrder_D64(mlib_d64 *xz, 
mlib_s32 n);

Description Each of these functions changes the encoding of each element from big endian to little endian, 
or from little endian to big endian.

It reverses the byte order of each element of the vector, in place.

Parameters Each of the functions takes the following arguments:

xz Pointer to input and output vector.

n Number of elements in the vectors.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
### `mlib_VectorReverseByteOrder_S16(3MLIB)`

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
`mlib_VectorReverseByteOrder(3MLIB)`, `mlib_VectorReverseByteOrder_Inp(3MLIB)`, `mlib_VectorReverseByteOrder_S16_S16(3MLIB)`, `attributes(5)`
Each of these functions changes the encoding of each element from big endian to little endian, or from little endian to big endian.

It copies and reverses the byte order of each element of the input vector into the output vector.

Each of the functions takes the following arguments:

- `z` Pointer to the output vector.
- `x` Pointer to input vector.
- `n` Number of elements in the vectors.

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:
### mlib_VectorReverseByteOrder_S16(3MLIB)

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VectorReverseByteOrder(3MLIB), mlib_VectorReverseByteOrder_Inp(3MLIB), mlib_VectorReverseByteOrder_S16(3MLIB), attributes(5)
Name mlib_VectorScale_U8_Mod, mlib_VectorScale_U8_Sat, mlib_VectorScale_U8C_Mod, mlib_VectorScale_U8C_Sat, mlib_VectorScale_S8_Mod, mlib_VectorScale_S8_Sat, mlib_VectorScale_S8C_Mod, mlib_VectorScale_S8C_Sat, mlib_VectorScale_S16_Mod, mlib_VectorScale_S16_Sat, mlib_VectorScale_S16C_Mod, mlib_VectorScale_S16C_Sat, mlib_VectorScale_S32_Mod, mlib_VectorScale_S32_Sat, mlib_VectorScale_S32C_Mod, mlib_VectorScale_S32C_Sat – vector linear scaling, in place

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorScale_U8_Mod(mlib_u8 *xz,
    const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_U8_Sat(mlib_u8 *xz,
    const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_U8C_Mod(mlib_u8 *xz,
    const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_U8C_Sat(mlib_u8 *xz,
    const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8_Mod(mlib_s8 *xz,
    const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8_Sat(mlib_s8 *xz,
    const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8C_Mod(mlib_s8 *xz,
    const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8C_Sat(mlib_s8 *xz,
    const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_Mod(mlib_s16 *xz,
    const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_Sat(mlib_s16 *xz,
    const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_Mod(mlib_s16 *xz,
    const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_Sat(mlib_s16 *xz,
    const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_Mod(mlib_s32 *xz,
    const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_Sat(mlib_s32 *xz,
    const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_Mod(mlib_s32 *xz,
    const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_Sat(mlib_s32 *xz, const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);

Description
Each of these functions performs an in-place scaling of a vector by multiplying by a scalar and adding an offset.

For real data, the following equation is used:

xz[i] = a[0]*xz[i] + b[0]

where i = 0, 1, ..., (n - 1).

For complex data, the following equation is used:

tmp = xz[2*i]
xz[2*i] = a[0]*tmp - a[1]*xz[2*i + 1] + b[0]
xz[2*i + 1] = a[1]*tmp + a[0]*xz[2*i + 1] + b[1]

where i = 0, 1, ..., (n - 1).

Parameters
Each of the functions takes the following arguments:

xz  Pointer to the first element of the source and destination vector.

a  Pointer to the source scaling factor. When the function is used with complex data types, a[0] contains the real part of the scaling factor, and a[1] contains the imaginary part of the scaling factor.

b  Pointer to the source offset. When the function is used with complex data types, b[0] contains the real part of the offset, and b[1] contains the imaginary part of the offset.

n  Number of elements in the vectors.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VectorScale_U8_U8_Mod(3MLIB), attributes(5)
mlib_VectorScale_U8_U8_Mod, mlib_VectorScale_U8_U8_Sat,
mlib_VectorScale_U8C_U8C_Mod, mlib_VectorScale_U8C_U8C_Sat,
mlib_VectorScale_S8_S8_Mod, mlib_VectorScale_S8_S8_Sat,
mlib_VectorScale_S8C_S8C_Mod, mlib_VectorScale_S8C_S8C_Sat,
mlib_VectorScale_S16_U8_Mod, mlib_VectorScale_S16_U8_Sat,
mlib_VectorScale_S16_S8_Mod, mlib_VectorScale_S16_S8_Sat,
mlib_VectorScale_S16_S16_Mod, mlib_VectorScale_S16_S16_Sat,
mlib_VectorScale_S16C_U8C_Mod, mlib_VectorScale_S16C_U8C_Sat,
mlib_VectorScale_S16C_S8C_Mod, mlib_VectorScale_S16C_S8C_Sat,
mlib_VectorScale_S16C_S16C_Mod, mlib_VectorScale_S16C_S16C_Sat,
mlib_VectorScale_S32_S16_Mod, mlib_VectorScale_S32_S16_Sat,
mlib_VectorScale_S32_S32_Mod, mlib_VectorScale_S32_S32_Sat–vector linear
scaling
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_VectorScale_U8_U8_Mod(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_U8_U8_Sat(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_U8C_U8C_Mod(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_U8C_U8C_Sat(mlib_u8 *z, const mlib_u8 *x,
const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8_S8_Mod(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8_S8_Sat(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8C_S8C_Mod(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S8C_S8C_Sat(mlib_s8 *z, const mlib_s8 *x,
const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x,
const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x,
const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x,
const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_S8_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_S16_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16_S16_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_U8C_Mod(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_U8C_Sat(mlib_s16 *z, const mlib_u8 *x, const mlib_u8 *a, const mlib_u8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_S8C_Mod(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_S8C_Sat(mlib_s16 *z, const mlib_s8 *x, const mlib_s8 *a, const mlib_s8 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_S16C_Mod(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S16C_S16C_Sat(mlib_s16 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_S16_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_S16_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_S32_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32_S32_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_S16C_Mod(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_S16C_Sat(mlib_s32 *z, const mlib_s16 *x, const mlib_s16 *a, const mlib_s16 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_S32C_Mod(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);
mlib_status mlib_VectorScale_S32C_S32C_Sat(mlib_s32 *z, const mlib_s32 *x, const mlib_s32 *a, const mlib_s32 *b, mlib_s32 n);

Description

Each of these functions scales a vector by multiplying by a scalar and adding an offset.

For real data, the following equation is used:

\[ z[i] = a[0] \times x[i] + b[0] \]
where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
z[2*i] &= a[0]*x[2*i] - a[1]*x[2*i + 1] + b[0] \\
z[2*i + 1] &= a[1]*x[2*i] + a[0]*x[2*i + 1] + b[1]
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the destination vector.
- \( x \) Pointer to the first element of the source vector.
- \( a \) Pointer to the source scaling factor. When the function is used with complex data types, \( a[0] \) contains the real part of the scaling factor, and \( a[1] \) contains the imaginary part of the scaling factor.
- \( b \) Pointer to the source offset. When the function is used with complex data types, \( b[0] \) contains the real part of the offset, and \( b[1] \) contains the imaginary part of the offset.
- \( n \) Number of elements in the vectors.

**Return Values** Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** mlib_VectorScale_U8_U8_Mod(3MLIB), attributes(5)
Each of these functions sets a vector to a specified value.

For real data, the following equation is used:

\[ z[i] = c[0] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[ z[2*i] = c[0] \]
\[ z[2*i + 1] = c[1] \]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters**

Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the destination vector.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scaling factor for the real part, and \( c[1] \) contains the scaling factor for the imaginary part.
- \( n \) Number of elements in the vector.
Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  attributes(5)
Name  mlib_VectorSplit_U8_U8C, mlib_VectorSplit_S8_S8C, mlib_VectorSplit_S16_S16C, mlib_VectorSplit_S32_S32C – vector split

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>
          mlib_status mlib_VectorSplit_U8_U8C(mlib_u8 *r, mlib_u8 *i,
            const mlib_u8 *x, mlib_s32 n);
          mlib_status mlib_VectorSplit_S8_S8C(mlib_s8 *r, mlib_s8 *i,
            const mlib_s8 *x, mlib_s32 n);
          mlib_status mlib_VectorSplit_S16_S16C(mlib_s16 *r, mlib_s16 *i,
            const mlib_s16 *x, mlib_s32 n);
          mlib_status mlib_VectorSplit_S32_S32C(mlib_s32 *r, mlib_s32 *i,
            const mlib_s32 *x, mlib_s32 n);

Description  Each of these functions splits a complex vector into separate vectors containing the real and imaginary parts.

The following equation is used:

r[k] = z[2k]
i[k] = z[2k + 1]

where k = 0, 1, ..., (n - 1).

Parameters  Each of the functions takes the following arguments:

r  Pointer to the first element of the real part.
i  Pointer to the first element of the imaginary part.
x  Pointer to the first complex element of the source vector. x[2k] contains the real part, and x[2k + 1] contains the imaginary part.
n  Number of elements in the vectors.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VectorMerge_U8C_U8(3MLIB), attributes(5)
Name  
mlib_VectorSubS_U8_Mod, mlib_VectorSubS_U8_Sat, mlib_VectorSubS_U8C_Mod, 
mlib_VectorSubS_U8C_Sat, mlib_VectorSubS_S8_Mod, mlib_VectorSubS_S8_Sat, 
mlib_VectorSubS_S8C_Mod, mlib_VectorSubS_S8C_Sat, mlib_VectorSubS_S16_Mod, 
mlib_VectorSubS_S16_Sat, mlib_VectorSubS_S16C_Mod, mlib_VectorSubS_S16C_Sat, 
mlib_VectorSubS_S32_Mod, mlib_VectorSubS_S32_Sat, mlib_VectorSubS_S32C_Mod, 
mlib_VectorSubS_S32C_Sat – vector subtraction from scalar, in place

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorSubS_U8_Mod(mlib_u8 *xz, 
       const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_U8_Sat(mlib_u8 *xz, 
       const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_U8C_Mod(mlib_u8 *xz, 
       const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_U8C_Sat(mlib_u8 *xz, 
       const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8_Mod(mlib_s8 *xz, 
       const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8_Sat(mlib_s8 *xz, 
       const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8C_Mod(mlib_s8 *xz, 
       const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8C_Sat(mlib_s8 *xz, 
       const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_Mod(mlib_s16 *xz, 
       const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_Sat(mlib_s16 *xz, 
       const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16C_Mod(mlib_s16 *xz, 
       const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16C_Sat(mlib_s16 *xz, 
       const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32_Mod(mlib_s32 *xz, 
       const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32_Sat(mlib_s32 *xz, 
       const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32C_Mod(mlib_s32 *xz, 
       const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32C_Sat(mlib_s32 *xz, 
       const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32C_Sat(mlib_s32 *xz,
  const mlib_s32 *c, mlib_s32 n);

Description Each of these functions performs an in-place subtraction of a vector from a scalar.

For real data, the following equation is used:

\[ xz[i] = c[0] - xz[i] \]

where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

Parameters Each of the functions takes the following arguments:

- \( xz \) Pointer to the first element of the source and destination vector.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( n \) Number of elements in the vectors.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VectorSubS_U8_U8_Mod(3MLIB), attributes(5)
Name  
mlib_VectorSubS_U8_U8_Mod, mlib_VectorSubS_U8_U8_Sat,
mlib_VectorSubS_U8C_U8C_Mod, mlib_VectorSubS_U8C_U8C_Sat,
mlib_VectorSubS_S8_S8_Mod, mlib_VectorSubS_S8_S8_Sat,
mlib_VectorSubS_S8C_S8C_Mod, mlib_VectorSubS_S8C_S8C_Sat,
mlib_VectorSubS_S16_U8_Mod, mlib_VectorSubS_S16_U8_Sat,
mlib_VectorSubS_S16_S8_Mod, mlib_VectorSubS_S16_S8_Sat,
mlib_VectorSubS_S16_S16_Mod, mlib_VectorSubS_S16_S16_Sat,
mlib_VectorSubS_S16C_U8C_Mod, mlib_VectorSubS_S16C_U8C_Sat,
mlib_VectorSubS_S16C_S8C_Mod, mlib_VectorSubS_S16C_S8C_Sat,
mlib_VectorSubS_S16C_S16C_Mod, mlib_VectorSubS_S16C_S16C_Sat,
mlib_VectorSubS_S32_S16_Mod, mlib_VectorSubS_S32_S16_Sat,
mlib_VectorSubS_S32_S32_Mod, mlib_VectorSubS_S32_S32_Sat,
mlib_VectorSubS_S32C_S16C_Mod, mlib_VectorSubS_S32C_S16C_Sat,
mlib_VectorSubS_S32C_S32C_Mod, mlib_VectorSubS_S32C_S32C_Sat – vector subtraction
from scalar

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorSubS_U8_U8_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_U8_U8_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_U8C_U8C_Mod(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_U8C_U8C_Sat(mlib_u8 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8_S8_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8_S8_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8C_S8C_Mod(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S8C_S8C_Sat(mlib_s8 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_U8_Mod(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_U8_Sat(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_S8_Mod(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_S8_Sat(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_S16_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16_S16_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16C_S16C_Mod(mlib_s16c *z,
    const mlib_s16c *x, const mlib_s16c *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S16C_S16C_Sat(mlib_s16c *z,
    const mlib_s16c *x, const mlib_s16c *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32_S16_Mod(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32_S16_Sat(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32_S32_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32_S32_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32C_S32C_Mod(mlib_s32c *z,
    const mlib_s32c *x, const mlib_s32c *c, mlib_s32 n);
mlib_status mlib_VectorSubS_S32C_S32C_Sat(mlib_s32c *z,
    const mlib_s32c *x, const mlib_s32c *c, mlib_s32 n);
Each of these functions subtracts a vector from a scalar.

For real data, the following equation is used:

\[ z[i] = c[0] - x[i] \]
where \( i = 0, 1, \ldots, (n - 1) \).

For complex data, the following equation is used:

\[
\begin{align*}
  z[2*i] &= c[0] - x[2*i] \\
\end{align*}
\]

where \( i = 0, 1, \ldots, (n - 1) \).

**Parameters** Each of the functions takes the following arguments:

- \( z \) Pointer to the first element of the destination vector.
- \( x \) Pointer to the first element of the source vector.
- \( c \) Pointer to the source scalar. When the function is used with complex data types, \( c[0] \) contains the scalar for the real part, and \( c[1] \) contains the scalar for the imaginary part.
- \( n \) Number of elements in the vectors.

**Return Values** Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

**Attributes** See attributes\(^{(5)}\) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** \texttt{mlib_VectorSubS_U8_U8_Mod(3MLIB)}, attributes\(^{(5)}\)
### mlib_VectorSub_U8_Mod(3MLIB)

**Name**
mlib_VectorSub_U8_Mod, mlib_VectorSub_U8_Sat, mlib_VectorSub_U8C_Mod,
mlib_VectorSub_U8C_Sat, mlib_VectorSub_S8_Mod, mlib_VectorSub_S8_Sat,
mlib_VectorSub_S8C_Mod, mlib_VectorSub_S8C_Sat, mlib_VectorSub_S16_Mod,
mlib_VectorSub_S16_Sat, mlib_VectorSub_S16C_Mod, mlib_VectorSub_S16C_Sat,
mlib_VectorSub_S32_Mod, mlib_VectorSub_S32_Sat, mlib_VectorSub_S32C_Mod,
mlib_VectorSub_S32C_Sat – vector subtraction, in place

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_VectorSub_U8_Mod(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_U8_Sat(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_U8C_Mod(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_U8C_Sat(mlib_u8 *xz,
    const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8_Mod(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8_Sat(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8C_Mod(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8C_Sat(mlib_s8 *xz,
    const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_Mod(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_Sat(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_Mod(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_Sat(mlib_s16 *xz,
    const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_Mod(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_Sat(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32C_Mod(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 n);
```
mlib_status mlib_VectorSub_S32C_Sat(mlib_s32 *xz,
    const mlib_s32 *y, mlib_s32 n);

Description Each of these functions performs an in-place subtraction of a vector from another vector.

It uses the following equation:

\[ xz[i] = xz[i] - y[i] \]

where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2*n - 1) \) for complex data.

Parameters Each of the functions takes the following arguments:

- \( xz \) Pointer to the first element of the first source and destination vector.
- \( y \) Pointer to the first element of the second source vector.
- \( n \) Number of elements in the vectors.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VectorSub_U8_U8_Mod(3MLIB), attributes(5)
**Synopsis**

cc [ flag... ] file... -lmlib [ library... ]

```
#include <mlib.h>

mlib_status mlib_VectorSub_U8_U8_Mod(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_U8_U8_Sat(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_U8C_U8C_Mod(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_U8C_U8C_Sat(mlib_u8 *z,
const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8_S8_Mod(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8_S8_Sat(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8C_S8C_Mod(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S8C_S8C_Sat(mlib_s8 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_S8_Mod(mlib_s16 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_S8_Sat(mlib_s16 *z,
const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_S16_Mod(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_S16_Sat(mlib_s16 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S16_Mod(mlib_s32 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S16_Sat(mlib_s32 *z,
const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S32_Mod(mlib_s32 *z,
const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S32_Sat(mlib_s32 *z,
const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
```

**Name**

mlib_VectorSub_U8_U8_Mod, mlib_VectorSub_U8_U8_Sat,
mlib_VectorSub_U8C_U8C_Mod, mlib_VectorSub_U8C_U8C_Sat,
mlib_VectorSub_S8_S8_Mod, mlib_VectorSub_S8_S8_Sat,
mlib_VectorSub_S8C_S8C_Mod, mlib_VectorSub_S8C_S8C_Sat,
mlib_VectorSub_S16_U8_Mod, mlib_VectorSub_S16_U8_Sat,
mlib_VectorSub_S16_S8_Mod, mlib_VectorSub_S16_S8_Sat,
mlib_VectorSub_S16_S16_Mod, mlib_VectorSub_S16_S16_Sat,
mlib_VectorSub_S16C_S16C_Mod, mlib_VectorSub_S16C_S16C_Sat,
mlib_VectorSub_S32_S16_Mod, mlib_VectorSub_S32_S16_Sat,
mlib_VectorSub_S32_S32_Mod, mlib_VectorSub_S32_S32_Sat,
mlib_VectorSub_S32C_S16C_Mod, mlib_VectorSub_S32C_S16C_Sat,
mlib_VectorSub_S32C_S32C_Mod, mlib_VectorSub_S32C_S32C_Sat – vector subtraction
mlib_status mlib_VectorSub_S16_S16_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16_S16_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_U8C_Mod(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_U8C_Sat(mlib_s16 *z,
    const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_S8C_Mod(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_S8C_Sat(mlib_s16 *z,
    const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_S16C_Mod(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S16C_S16C_Sat(mlib_s16 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S16_Mod(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S16_Sat(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S32_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32_S32_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32C_S16C_Mod(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32C_S16C_Sat(mlib_s32 *z,
    const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32C_S32C_Mod(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
mlib_status mlib_VectorSub_S32C_S32C_Sat(mlib_s32 *z,
    const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);

Description  Each of these functions subtracts one vector from another vector.
          It uses the following equation:
          \[ z[i] = x[i] - y[i] \]
          where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2n - 1) \) for complex data.

Parameters  Each of the functions takes the following arguments:
Pointers to the first element of the destination vector.
Pointers to the first element of the first source vector.
Pointers to the first element of the second source vector.
Number of elements in the vectors.

Return Values: Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_VectorSub_U8_U8_Mod(3MLIB), attributes(5)
mlib_VectorSumAbsDiff_U8_Sat, mlib_VectorSumAbsDiff_S8_Sat, mlib_VectorSumAbsDiff_S16_Sat, mlib_VectorSumAbsDiff_S32_Sat – sum of the absolute values of the differences of two vectors

### Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorSumAbsDiff_U8_Sat(mlib_d64 *z,
        const mlib_u8 *x, const mlib_u8 *y, mlib_s32 n);
mlib_status mlib_VectorSumAbsDiff_S8_Sat(mlib_d64 *z,
        const mlib_s8 *x, const mlib_s8 *y, mlib_s32 n);
mlib_status mlib_VectorSumAbsDiff_S16_Sat(mlib_d64 *z,
        const mlib_s16 *x, const mlib_s16 *y, mlib_s32 n);
mlib_status mlib_VectorSumAbsDiff_S32_Sat(mlib_d64 *z,
        const mlib_s32 *x, const mlib_s32 *y, mlib_s32 n);
```

### Description

Each of these functions computes the sum of the absolute values of the differences of two vectors.

The following equation is used:

\[
    z[0] = \sum_{i=0}^{n-1} |x[i] - y[i]|
\]

### Parameters

Each of the functions takes the following arguments:

- **z** Pointer to the sum of the absolute differences between two vectors.
- **x** Pointer to the first element of the first source vector.
- **y** Pointer to the first element of the second source vector.
- **n** Number of elements in the vectors.

### Return Values

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

attributes(5)
Name mlib_VectorSumAbs_U8_Sat, mlib_VectorSumAbs_S8_Sat, mlib_VectorSumAbs_S16_Sat, mlib_VectorSumAbs_S32_Sat – sum of the absolute values of a vector

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VectorSumAbs_U8_Sat(mlib_d64 *z, const mlib_u8 *x,
                        mlib_s32 n);
mlib_status mlib_VectorSumAbs_S8_Sat(mlib_d64 *z, const mlib_s8 *x,
                        mlib_s32 n);
mlib_status mlib_VectorSumAbs_S16_Sat(mlib_d64 *z, const mlib_s16 *x,
                        mlib_s32 n);
mlib_status mlib_VectorSumAbs_S32_Sat(mlib_d64 *z, const mlib_s32 *x,
                        mlib_s32 n);

Description Each of these functions computes the sum of the absolute values of a vector.

The following equation is used:

\[
\begin{align*}
n-1 \\
z[0] &= \sum_{i=0}^{n-1} |x[i]|
\end{align*}
\]

Parameters Each of the functions takes the following arguments:

- \( z \) Pointer to the sum of the absolute values of the vector.
- \( x \) Pointer to the first element of the first source vector.
- \( n \) Number of elements in the vectors.

Return Values Each of the functions returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also attributes(5)
Each of these functions initializes a vector to zero. The following equation is used:
\[ z[i] = 0 \]
where \( i = 0, 1, \ldots, (n - 1) \) for real data; \( i = 0, 1, \ldots, (2*n - 1) \) for complex data.

**Parameters**
Each of the functions takes the following arguments:
- \( z \) Pointer to the first element of the destination vector.
- \( n \) Number of elements in the vector.

**Return Values**
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
attributes(5)
The `mlib_version()` function returns a string about the version of the library being used.

This function returns a string in the following format:

```
lib_name:version:build_date:target_isa
```

The `lib_name` is `mediaLib`. The `version` consists of four digits. The first two digits of the version are the major version. The third digit is the minor version, and the fourth digit is the micro version. The `build_date` is in the `yyyymmdd` format. The `target_isa` is the value used for the `-xarch=a` flag of the compiler when the library was built. For example, the following version string corresponds to a library in `mediaLib` version 2.1.0, which was built on 11/01/2001 and for the `sparcv8plus+vis` architecture.

```
mediaLib:0210:20011101:v8plusa
```

**Parameters**
The function takes no argument.

**Return Values**
The function returns a pointer to a string of characters.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`attributes(5)`
mlib_VideoAddBlock_U8_S16 – adds motion-compensated 8x8 block to the current block

Synopsis

cc [ flag... ] file... -tclib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoAddBlock_U8_S16(mlib_u8 *curr_block,
const mlib_s16 *mc_block, mlib_s32 stride);

Description

The mlib_VideoAddBlock_U8_S16() function performs additions of prediction and
coefficient data. In other words, the function adds a motion-compensated 8x8 block to the
current block. The stride applies to the current block.

Parameters

The function takes the following arguments:

curr_block  Pointer to the current block. curr_block must be 8-byte aligned.
mc_block  Pointer to an 8x8 motion-compensated block (prediction data). mc_block
must be 8-byte aligned.
stride  Stride, in bytes, between adjacent rows in the current block. stride must be
a multiple of eight.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_VideoAddBlock_U8_S16(3MLIB),
mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263Decimate_U8_U8(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpX_U8_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
The `mlib_VideoColorABGR2JFIFYCC420()` function performs color space conversion from ABGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

### Parameters
The function takes the following arguments:

- `y0` Pointer to upper destination Y component row. `y0` must be 8-byte aligned.
- `y1` Pointer to lower destination Y component row. `y1` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `abgr0` Pointer to upper source ABGR multi-component row. `abgr0` must be 8-byte aligned.
- `abgr1` Pointer to lower source ABGR multi-component row. `abgr1` must be 8-byte aligned.
- `n` Length of Y component row. `n` must be even. The length of Cb and Cr component rows must be `n/2`. The length of the ABGR multi-component row must be `4*n`.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_VideoColorABGR2JFIFYCC422(3MLIB)`, `mlib_VideoColorARGB2JFIFYCC420(3MLIB)`, `mlib_VideoColorARGB2JFIFYCC422(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC420(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC422(3MLIB)`, `attributes(5)`
The `mlib_VideoColorABGR2JFIFYCC422()` function performs color space conversion from ABGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters**
- **y**: Pointer to destination Y component row. y must be 8-byte aligned.
- **cb**: Pointer to destinationCb component row. cb must be 8-byte aligned.
- **cr**: Pointer to destination Cr component row. cr must be 8-byte aligned.
- **abgr**: Pointer to source ABGR multi-component row. abgr must be 8-byte aligned.
- **n**: Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the ABGR multi-component row must be 4*n.

**Return Values**
- The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoColorABGR2JFIFYCC420(3MLIB)`
- `mlib_VideoColorARGB2JFIFYCC420(3MLIB)`
- `mlib_VideoColorARGB2JFIFYCC422(3MLIB)`
- `mlib_VideoColorRGB2JFIFYCC420(3MLIB)`
- `mlib_VideoColorRGB2JFIFYCC422(3MLIB)`
- attributes(5)
mlib_VideoColorABGR2JFIFYCC444() - ABGR to JFIF YCbCr color conversion

**Synopsis**

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_VideoColorABGR2JFIFYCC444(mlib_u8 *y, mlib_u8 *cb,
                                          mlib_u8 *cr, const mlib_u8 *abgr,
                                          mlib_s32 n);
```

**Description**
The `mlib_VideoColorABGR2JFIFYCC444()` function performs color space conversion from ABGR to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

- \( Y = 0.2990 \times R + 0.5870 \times G + 0.1140 \times B \)
- \( Cb = -0.16874 \times R - 0.33126 \times G + 0.50000 \times B + 128 \)
- \( Cr = 0.50000 \times R - 0.41869 \times G - 0.08131 \times B + 128 \)

**Parameters**
The function takes the following arguments:

- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination `Cb` component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination `Cr` component row. `cr` must be 8-byte aligned.
- `abgr` Pointer to source ABGR multi-component row. `abgr` must be 8-byte aligned.
- `n` Length of `Y` component row. The length of `Cb` and `Cr` component rows must be `n`. The length of the ABGR multi-component row must be 4*n.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoColorARGB2JFIFYCC444(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC444(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC444_S16(3MLIB)`, attributes(5)
REFERENCE

Multimedia Library Functions - Part 7
The `mlib_VideoColorABGR2RGB()` function performs ABGR to RGB color order conversion.

The function takes the following arguments:

- `rgb` Pointer to RGB row.
- `abgr` Pointer to ABGR row.
- `n` Number of pixels.

The function returns `MLIB_SUCCESS` if successful. Otherwise, it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoColorARGB2RGB(3MLIB)`
- `mlib_VideoColorRGB2ABGR(3MLIB)`
- `mlib_VideoColorRGB2ARGB(3MLIB)`
- attributes(5)
Name: mlib_VideoColorABGRint_to_ARGBint – convert ABGR interleaved to ARGB

Synopsis:
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
void mlib_VideoColorABGRint_to_ARGBint(mlib_u32 * ARGB,
    const mlib_u32 * ABGR, mlib_s32 w, mlib_s32 h, mlib_s32 dlb,
    mlib_s32 slb);
```

Description: The ABGR pixel stream is broken apart and recombined into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The buffers have dimensions w and h. Within each 32-bit input word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), R (bits 23-16), G (bits 15-8), and B (bits 7-0).

Parameters: The function takes the following arguments:
- **ARGB**: Pointer to output buffer.
- **ABGR**: Pointer to input buffer.
- **w**: Image width in pixels.
- **h**: Image height in lines.
- **dlb**: Linebytes for output buffer.
- **slb**: Linebytes for input buffer.

Return Values: None.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_VideoColorRGBAint_to_ABGRint(3MLIB),
mlib_VideoColorBGRAint_to_ABGRint(3MLIB), attributes(5)
mlib_VideoColorARGB2JFIFYCC420(3MLIB)

Name  mlib_VideoColorARGB2JFIFYCC420 – ARGB to JFIF YCbCr color conversion

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorARGB2JFIFYCC420(mlib_u8 *y0,
    mlib_u8 *y1, mlib_u8 *cb, mlib_u8 *cr, const mlib_u8 *argb0,
    const mlib_u8 *argb1, mlib_s32 n);

Description  The mlib_VideoColorARGB2JFIFYCC420() function performs color space conversion from ARGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

Parameters  The function takes the following arguments:

- y0  Pointer to upper destination Y component row. y0 must be 8-byte aligned.
- y1  Pointer to lower destination Y component row. y1 must be 8-byte aligned.
- cb  Pointer to destination Cb component row. cb must be 8-byte aligned.
- cr  Pointer to destination Cr component row. cr must be 8-byte aligned.
- argb0  Pointer to upper source ARGB multi-component row. argb0 must be 8-byte aligned.
- argb1  Pointer to lower source ARGB multi-component row. argb1 must be 8-byte aligned.
- n  Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the ARGB multi-component row must be 4*n.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorABGR2JFIFYCC420(3MLIB), mlib_VideoColorABGR2JFIFYCC422(3MLIB), mlib_VideoColorARGB2JFIFYCC420(3MLIB), mlib_VideoColorARGB2JFIFYCC422(3MLIB), mlib_VideoColorRGBA2JFIFYCC420(3MLIB), mlib_VideoColorRGBA2JFIFYCC422(3MLIB), attributes(5)
The `mlib_VideoColorARGB2JFIFYCC422()` function performs color space conversion from ARGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters**
- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `argb` Pointer to source ARGB multi-component row. `argb` must be 8-byte aligned.
- `n` Length of Y component row. `n` must be even. The length of Cb and Cr component rows must be `n/2`. The length of the ARGB multi-component row must be 4*n.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoColorABGR2JFIFYCC420(3MLIB)`, `mlib_VideoColorABGR2JFIFYCC422(3MLIB)`
- `mlib_VideoColorARGB2JFIFYCC420(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC420(3MLIB)`
The mlib_VideoColorARGB2JFIFYCC444() function performs color space conversion from ARGB to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
\begin{align*}
Y &= 0.2990 \times R + 0.5870 \times G + 0.1140 \times B \\
Cb &= -0.16874 \times R - 0.33126 \times G + 0.50000 \times B + 128 \\
Cr &= 0.50000 \times R - 0.41869 \times G - 0.08131 \times B + 128
\end{align*}
\]

The function takes the following arguments:

- `y`: Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb`: Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr`: Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `argb`: Pointer to source ARGB multi-component row. `rgb` must be 8-byte aligned.
- `n`: Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the ARGB multi-component row must be `4*n`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorARGB2JFIFYCC444(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC444(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC444_S16(3MLIB)`, `attributes(5)`
The `mlib_VideoColorARGB2RGB()` function performs ARGB to RGB color order conversion.

### Parameters
- `rgb`: Pointer to RGB row.
- `argb`: Pointer to ARGB row.
- `n`: Number of pixels.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_VideoColorABGR2RGB(3MLIB)`, `mlib_VideoColorRGB2ABGR(3MLIB)`, `mlib_VideoColorRGB2ARGB(3MLIB)`, attributes(5)
# mlib_VideoColorBGR2JFIFYCC420

The `mlib_VideoColorBGR2JFIFYCC420()` function performs color space conversion from BGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

## Parameters

- `y0` Pointer to upper destination Y component row. `y0` must be 8-byte aligned.
- `y1` Pointer to lower destination Y component row. `y1` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `bgr0` Pointer to upper source BGR multi-component row. `bgr0` must be 8-byte aligned.
- `bgr1` Pointer to lower source BGR multi-component row. `bgr1` must be 8-byte aligned.
- `n` Length of Y component row. `n` must be even. The length of Cb and Cr component rows must be `n/2`. The length of the BGR multi-component row must be `3*n`.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoColorBGR2JFIFYCC422(3MLIB), mlib_VideoColorBGR2JFIFYCC444(3MLIB), mlib_VideoColorBGR2JFIFYCC444_S16(3MLIB), attributes(5)`
mlib_VideoColorBGR2JFIFYCC422

**Name**
mlib_VideoColorBGR2JFIFYCC422 – BGR to JFIF YCbCr color conversion

**Synopsis**
c
flag... file... -lmlib [ library... ]

#include <mlib.h>

```c
mlib_status mlib_VideoColorBGR2JFIFYCC422(mlib_u8 *y, mlib_u8 *cb,
mlib_u8 *cr, const mlib_u8 *bgr, mlib_s32 n);
```

**Description**
The `mlib_VideoColorBGR2JFIFYCC422` function performs color space conversion from BGR to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters**
The function takes the following arguments:

- `y` Pointer to destination Y component row. y must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. cb must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. cr must be 8-byte aligned.
- `bgr` Pointer to source BGR multi-component row. bgr must be 8-byte aligned.
- `n` Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the BGR multi-component row must be 3*n.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
The `mlib_VideoColorBGR2JFIFYCC444()` function performs color space conversion from BGR to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
\begin{align*}
Y &= 0.2990 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \\
Cb &= -0.16874 \cdot R - 0.33126 \cdot G + 0.50000 \cdot B + 128 \\
Cr &= 0.50000 \cdot R - 0.41869 \cdot G - 0.08131 \cdot B + 128
\end{align*}
\]

The function takes the following arguments:

- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `bgr` Pointer to source BGR multi-component row. `bgr` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the BGR multi-component row must be 3*n.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_VideoColorBGR2JFIFYCC420(3MLIB), mlib_VideoColorBGR2JFIFYCC422(3MLIB), mlib_VideoColorBGR2JFIFYCC444_S16(3MLIB), attributes(5)`
The `mlib_VideoColorBGR2JFIFYCC444_S16()` function performs color space conversion from BGR to YCbCr when used in the JPEG File Interchange Format (JFIF).

Both the input BGR components and the output YCbCr components are supposed to be in the range of [0, 4095].

The following equation is used:

\[
\begin{align*}
Y &= 0.29900 \times R + 0.58700 \times G + 0.11400 \times B \\
Cb &= -0.16874 \times R - 0.33126 \times G + 0.50000 \times B + 2048 \\
Cr &= 0.50000 \times R - 0.41869 \times G - 0.08131 \times B + 2048
\end{align*}
\]

The function takes the following arguments:

- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `bgr` Pointer to source BGR multi-component row. `bgr` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the BGR multi-component row must be `3*n`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoColorBGR2JFIFYCC420(3MLIB)`, `mlib_VideoColorBGR2JFIFYCC422(3MLIB)`, `mlib_VideoColorBGR2JFIFYCC444(3MLIB)`, attributes(5)
mlib_VideoColorBGRAint_to_ABGRint(3MLIB)

**Name**
mlib_VideoColorBGRAint_to_ABGRint – convert BGRA interleaved to ABGR

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorBGRAint_to_ABGRint(mlib_u32 *ABGR,
    const mlib_u32 *BGRA, mlib_s32 w, mlib_s32 h, mlib_s32 dlb,
    mlib_s32 slb);
```

**Description**
The BGRA pixel stream is broken apart and recombined into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The buffers have dimensions \( w \) and \( h \). Within each 32-bit input word, the component ordering is \( B \) (bits 31-24), \( G \) (bits 23-16), \( R \) (bits 15-8), and \( A \) (bits 7-0). Within each 32-bit output word, the component ordering is \( A \) (bits 31-24), \( B \) (bits 23-16), \( G \) (bits 15-8), and \( R \) (bits 7-0).

**Parameters**
The function takes the following arguments:
- **ABGR** — Pointer to output buffer.
- **BGRA** — Pointer to input buffer.
- **w** — Image width in pixels.
- **h** — Image height in lines.
- **dlb** — Linebytes for output buffer.
- **slb** — Linebytes for input buffer.

**Return Values**
None.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_VideoColorABGRint_to_ARGBint(3MLIB),
mlib_VideoColorRGBAint_to_ABGRint(3MLIB), attributes(5)
Name  
mlib_VideoColorBGRint_to_ABGRint – convert BGR interleaved to ABGR interleaved

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorBGRint_to_ABGRint(mlib_u32 *ABGR,
    const mlib_u8 *BGR, const mlib_u8 *A_array, mlib_u8 A_const,
    mlib_s32 w, mlib_s32 h, mlib_s32 dl, mlib_s32 sl, mlib_s32 a);

Description  
The interleaved BGR stream, and the A values are combined into an A, B, G, R interleaved byte stream. Within each 24-bit input pixel, the component ordering is B (bits 23-16), G (bits 15-8), and R (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

The alpha values for this function work in the following fashion:

- If A_array pointer is not NULL, the values are taken from there. It has to have the same dimensions as the R, G, and B buffers.
- If A_array pointer is NULL, the alpha values for every pixel are set to A_const.

Parameters  
The function takes the following arguments:

- ABGR  Pointer to output buffer.
- BGR  Pointer to input buffer.
- A_array  Array of alpha values.
- A_const  Constant alpha value.
- w  Image width in pixels.
- h  Image height in lines.
- dl  Linebytes for output buffer.
- sl  Linebytes for input buffer.
- a  Linebytes for alpha buffer.

Return Values  
None.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_VideoColorGRBseq_to_ABGRint(3MLIB),
mlib_VideoColorRGBint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ARGBint(3MLIB),
mlib_VideoColorXRGBint_to_ABGRint(3MLIB),
mlib_VideoColorXRGBint_to_ARGBint(3MLIB), attributes(5)
The functions use the following equation for blending images:

\[ \text{dst} = (\text{src1} \times \text{src1}\_blend) + (\text{src2} \times \text{src2}\_blend) \]

The two multi-banded source images (src1 and src2) are blended together and stored in the destination image (dst). The image buffers pointed to by dst, src1, and src2 contain 4-banded ABGR images, 8 bits per component. src1\_w and src1\_h are the dimensions of the src1 input buffer. src2\_w and src2\_h are the dimensions of the src2 input buffer. The output buffer must be at least as large as the src1 input buffer. src2\_x and src2\_y are the offset of the src2 input buffer relative to src1. Where pixels in src2 overlap pixels in src1, the pixels are blended. Pixels in src1 which are outside of src2 are copied into dst. Pixels in the dst image outside of src1 are left unchanged. src1\_blend specifies the blend function to be applied to the pixels of src1 image and src2\_blend specifies the blend function to be applied to the pixels of src2.

Possible blend functions are:

- MLIB_BLEND_ZERO
- MLIB_BLEND_ONE
- MLIB_BLEND_SRC_ALPHA
- MLIB_BLEND_ONE_MINUS_SRC_ALPHA
- MLIB_BLEND_DST_ALPHA
- MLIB_BLEND_ONE_MINUS_DST_ALPHA

MLIB_BLEND_SRC_ALPHA is the alpha component of image src2 scaled to the range 0.0 to 1.0. MLIB_BLEND_DST_ALPHA is the alpha component of image src1 scaled to the range 0.0 to 1.0. All pixel components are treated as unsigned 8-bit quantities and the output pixel component values are clamped to the range 0 to 255.
For the `mlib_VideoColorBlendABGR_ResetAlpha()` function, the alpha value of every pixel in destination image is set to 0 after blending is complete.

**Parameters**

Each of the functions takes the following arguments:

- `dst` Pointer to output image.
- `src1` Pointer to 1st input image.
- `src2` Pointer to 2nd input image.
- `src1_w` `src1` image width in pixels.
- `src1_h` `src1` image height in rows.
- `src2_w` `src2` image width in pixels.
- `src2_h` `src2` image height in rows.
- `src2_x` `src2` horizontal displacement (in pixels), relative to the upper-left corner of `src1`.
- `src2_y` `src2` vertical displacement (in rows), relative to the upper-left corner of `src1`.
- `dst_lb` Linebytes for output image.
- `src1_lb` Linebytes for 1st input image.
- `src2_lb` Linebytes for 2nd input image.
- `src1_blend` Blend function for `src1` image.
- `src2_blend` Blend function for `src2` image.

**Return Values**

None.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_VideoColorBlendABGR_Inp(3MLIB), attributes(5)`
The functions use the following equation for blending images:

\[
src1dst = (src1dst * src1dst\_blend) + (src2 * src2\_blend)
\]

The two multi-banded source images (\(src1dst\) and \(src2\)) are blended together and the result is stored in \(src1dst\). \(src1dst\_blend\) specifies the blend function to be applied to the \(src1dst\) image and \(src2\_blend\) specifies the blend function to be applied to the \(src2\) image. \(src2\_x\) and \(src2\_y\) specify position of \(src2\) relative to the upper-left corner of \(src1dst\). \(src2\) is clipped to the boundaries of \(src1dst\), if needed.

Possible blend functions are:

- **MLIB_BLEND_ZERO**
- **MLIB_BLEND_ONE**
- **MLIB_BLEND_SRC_ALPHA**
- **MLIB_BLEND_ONE_MINUS_SRC_ALPHA**
- **MLIB_BLEND_DST_ALPHA**
- **MLIB_BLEND_ONE_MINUS_DST_ALPHA**

**MLIB_BLEND_DST_ALPHA** is the alpha band of image \(src1\) scaled to the range 0 to 1. **MLIB_BLEND_SRC_ALPHA** is the alpha band of image \(src2\) scaled to the range 0 to 1. The output pixel bands are clamped to the range 0 to 255.

For the **mlib_VideoColorBlendABGR_ResetAlpha_Inp()** function, the alpha value of every pixel in destination image is set to 0 after blending is complete.
Each of the functions takes the following arguments:

- `src1dst`: Pointer to 1st input image (also dest. image).
- `src2`: Pointer to 2nd input image.
- `src1dst_w`: `src1dst` image width in pixels.
- `src1dst_h`: `src1dst` image height in rows.
- `src2_w`: `src2` image width in pixels.
- `src2_h`: `src2` image height in rows.
- `src2_x`: `src2` horizontal displacement (in pixels), relative to the upper-left corner of `src1dst`.
- `src2_y`: `src2` vertical displacement (in rows), relative to the upper-left corner of `src1dst`.
- `src1dst_lb`: Linebytes for `src1dst` image.
- `src2_lb`: Linebytes for `src2` image.
- `src1dst_blend`: Blend function for `src1dst` image.
- `src2_blend`: Blend function for `src2` image.

None.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoColorBlendABGR(3MLIB), attributes(5)`
The `mlib_VideoColorCMYK2JFIFYCCK444()` function performs color space conversion from CMYK to YCbCrK when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
\begin{align*}
R &= (255 - C) \\
G &= (255 - M) \\
B &= (255 - Y) \\
Y &= 0.29900 \times R + 0.58700 \times G + 0.11400 \times B \\
Cb &= -0.16874 \times R - 0.33126 \times G + 0.50000 \times B + 128 \\
Cr &= 0.50000 \times R - 0.41869 \times G - 0.08131 \times B + 128 \\
K &= K
\end{align*}
\]

The function takes the following arguments:

- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `k` Pointer to destination K component row. `k` must be 8-byte aligned.
- `cmyk` Pointer to source CMYK multi-component row. `cmyk` must be 8-byte aligned.
- `n` Length of Y, Cb, Cr, and K component rows. The length of the CMYK multi-component row must be 4*n.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoColorJFIFYCCK2CMYK444(3MLIB), attributes(5)`
The `mlib_VideoColorJFIFYCC2ABGR444()` function performs color space conversion from YCbCr to ABGR when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
\begin{align*}
A &= 0xFF \\
R &= Y + 1.40200 \times (Cr - 128) \\
G &= Y - 0.34414 \times (Cb - 128) - 0.71414 \times (Cr - 128) \\
B &= Y + 1.77200 \times (Cb - 128)
\end{align*}
\]

The function takes the following arguments:

- `abgr` Pointer to destination ABGR multi-component row. `abgr` must be 8-byte aligned.
- `y` Pointer to source Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to source Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to source Cr component row. `cr` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the ABGR multi-component row must be `4*n`.

Return Values The function returns `MLIB_SUCCESS` if successful. Otherwise, it returns `MLIB_FAILURE`.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoColorJFIFYCC2ARGB444(3MLIB)`, `mlib_VideoColorJFIFYCC2RGB444(3MLIB)`, `mlib_VideoColorJFIFYCC2RGB444_S16(3MLIB)`, attributes(5)
The `mlib_VideoColorJFIFYCC2ARGB444()` function performs color space conversion from YCbCr to ARGB when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
\begin{align*}
A &= 0xFF \\
R &= Y + 1.40200 \times (Cr - 128) \\
G &= Y - 0.34414 \times (Cb - 128) - 0.71414 \times (Cr - 128) \\
B &= Y + 1.77200 \times (Cb - 128)
\end{align*}
\]

The function takes the following arguments:

- `argb` Pointer to destination ARGB multi-component row. `argb` must be 8-byte aligned.
- `y` Pointer to source Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to source Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to source Cr component row. `cr` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the ARGB multi-component row must be `4*n`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorJFIFYCC2ABGR444()` and `mlib_VideoColorJFIFYCC2RGB444_S16()`.

---

**Name**  
`mlib_VideoColorJFIFYCC2ARGB444` - JFIF YCbCr to ARGB color conversion

**Synopsis**  
```
c [ flag... ] file... -lmlib [ library... ]
```

```
#include <mlib.h>

mlib_status mlib_VideoColorJFIFYCC2ARGB444(mlib_u8 *argb, const mlib_u8 *y, const mlib_u8 *cb, const mlib_u8 *cr, mlib_s32 n);
```

**Description**  
The `mlib_VideoColorJFIFYCC2ARGB444()` function performs color space conversion from YCbCr to ARGB when used in the JPEG File Interchange Format (JFIF).

**Parameters**  
The function takes the following arguments:

- `argb` Pointer to destination ARGB multi-component row. `argb` must be 8-byte aligned.
- `y` Pointer to source Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to source Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to source Cr component row. `cr` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the ARGB multi-component row must be `4*n`.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
`mlib_VideoColorJFIFYCC2ABGR444(3MLIB)`, `mlib_VideoColorJFIFYCC2RGB444(3MLIB)`, `mlib_VideoColorJFIFYCC2RGB444_S16(3MLIB)`, attributes(5)
The `mlib_VideoColorJFIFYCC2RGB420()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

The function takes the following arguments:

- `rgb0` Pointer to upper destination RGB multi-component row. `rgb0` must be 8-byte aligned.
- `rgb1` Pointer to lower destination RGB multi-component row. `rgb1` must be 8-byte aligned.
- `y0` Pointer to upper destination Y component row. `y0` must be 8-byte aligned.
- `y1` Pointer to lower destination Y component row. `y1` must be 8-byte aligned.
- `cb0` Pointer to source upper Cb component row. `cb0` must be 8-byte aligned.
- `cr0` Pointer to source upper Cr component row. `cr0` must be 8-byte aligned.
- `cb1` Pointer to source middle Cb component row. `cb1` must be 8-byte aligned.
- `cr1` Pointer to source middle Cr component row. `cr1` must be 8-byte aligned.
- `cb2` Pointer to source lower Cb component row. `cb2` must be 8-byte aligned.
- `cr2` Pointer to source lower Cr component row. `cr2` must be 8-byte aligned.
- `n` Length of Y component row. `n` must be even. The length of Cb and Cr component rows must be `n/2`. The length of the RGB multi-component row must be `3*n`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_VideoColorJFIFYCC2RGB420_Nearest(3MLIB),  
mlib_VideoColorJFIFYCC2RGB422(3MLIB),  
mlib_VideoColorJFIFYCC2RGB422_Nearest(3MLIB), attributes(5)
mlib_VideoColorJFIFYCC2RGB420_Nearest

Name  mlib_VideoColorJFIFYCC2RGB420_Nearest – JFIF YCbCr to RGB color conversion

Synopsis  cc [ flag... ] file... -mlib [ library... ]

    #include <mlib.h>

    mlib_status mlib_VideoColorJFIFYCC2RGB420_Nearest(mlib_u8 *rgb0,
            mlib_u8 *rgb1, const mlib_u8 *y0,
            const mlib_u8 *y1, const mlib_u8 *cb,
            const mlib_u8 *cr, mlib_s32 n);

Description  The mlib_VideoColorJFIFYCC2RGB420_Nearest() function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

Parameters  The function takes the following arguments:

    rgb0   Pointer to upper destination RGB multi-component row. rgb0 must be 8-byte aligned.

    rgb1   Pointer to lower destination RGB multi-component row. rgb1 must be 8-byte aligned.

    y0     Pointer to upper destination Y component row. y0 must be 8-byte aligned.

    y1     Pointer to lower destination Y component row. y1 must be 8-byte aligned.

    cb     Pointer to source Cb component row. cb must be 8-byte aligned.

    cr     Pointer to source Cr component row. cr must be 8-byte aligned.

    n      Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the RGB multi-component row must be 3*n.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorJFIFYCC2RGB420(3MLIB), mlib_VideoColorJFIFYCC2RGB422(3MLIB), mlib_VideoColorJFIFYCC2RGB422_Nearest(3MLIB), attributes(5)
**Name**
mlib_VideoColorJFIFYCC2RGB422 – JFIF YCbCr to RGB color conversion

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_VideoColorJFIFYCC2RGB422(mlib_u8 *rgb,
    const mlib_u8 *y, const mlib_u8 *cb,
    const mlib_u8 *cr, mlib_s32 n);
```

**Description**
The `mlib_VideoColorJFIFYCC2RGB422()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters**
The function takes the following arguments:
- `rgb` Pointer to destination RGB multi-component row. rgb must be 8-byte aligned.
- `y` Pointer to destination Y component row. y must be 8-byte aligned.
- `cb` Pointer to source Cb component row. cb must be 8-byte aligned.
- `cr` Pointer to source Cr component row. cr must be 8-byte aligned.
- `n` Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the RGB multi-component row must be 3*n.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoColorJFIFYCC2RGB420(3MLIB)`
- `mlib_VideoColorJFIFYCC2RGB420_Nearest(3MLIB)`
- `mlib_VideoColorJFIFYCC2RGB422_Nearest(3MLIB), attributes(5)`
The `mlib_VideoColorJFIFYCC2RGB422_Nearest()` function performs color space conversion from YCbCr to RGB together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

**Parameters**

- `rgb` Pointer to destination RGB multi-component row. rgb must be 8-byte aligned.
- `y` Pointer to destination Y component row. y must be 8-byte aligned.
- `cb` Pointer to source Cb component row. cb must be 8-byte aligned.
- `cr` Pointer to source Cr component row. cr must be 8-byte aligned.
- `n` Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the RGB multi-component row must be 3*n.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoColorJFIFYCC2RGB420(3MLIB)`,
- `mlib_VideoColorJFIFYCC2RGB420_Nearest(3MLIB)`,
- `mlib_VideoColorJFIFYCC2RGB422(3MLIB)`, attributes(5)
Name  mlib_VideoColorJFIFYCC2RGB444 – JFIF YCbCr to RGB color conversion

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorJFIFYCC2RGB444(mlib_u8 *rgb, const mlib_u8 *y,
                                           const mlib_u8 *cb, const mlib_u8 *cr,
                                           mlib_s32 n);

Description  The mlib_VideoColorJFIFYCC2RGB444() function performs color space conversion from
YCbCr to RGB when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

R = Y + 1.40200 * (Cr - 128)
G = Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128)
B = Y + 1.77200 * (Cb - 128)

Parameters  The function takes the following arguments:

rgb  Pointer to destination RGB multi-component row. rgb must be 8-byte aligned.
y  Pointer to source Y component row. y must be 8-byte aligned.
cb  Pointer to source Cb component row. cb must be 8-byte aligned.
cr  Pointer to source Cr component row. cr must be 8-byte aligned.
n  Length of Y component row. The length of Cb and Cr component rows must be n. The
length of the RGB multi-component row must be 3*n.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorJFIFYCC2ABGR444(3MLIB), mlib_VideoColorJFIFYCC2ARGB444(3MLIB),
mlib_VideoColorJFIFYCC2RGB444_S16(3MLIB), attributes(5)
The `mlib_VideoColorJFIFYCC2RGB444_S16()` function performs color space conversion from YCbCr to RGB when used in the JPEG File Interchange Format (JFIF).

Both the input YCbCr components and the output RGB components are supposed to be in the range of [0, 4095].

The following equation is used:

\[
R = Y + 1.40200 \times (Cr - 2048) \\
G = Y - 0.34414 \times (Cb - 2048) - 0.71414 \times (Cr - 2048) \\
B = Y + 1.77200 \times (Cb - 2048)
\]

The function takes the following arguments:

- `rgb` Pointer to destination RGB multi-component row. rgb must be 8-byte aligned.
- `y` Pointer to source Y component row. y must be 8-byte aligned.
- `cb` Pointer to source Cb component row. cb must be 8-byte aligned.
- `cr` Pointer to source Cr component row. cr must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be n. The length of the RGB multi-component row must be 3*n.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorJFIFYCC2ABGR444(3MLIB), mlib_VideoColorJFIFYCC2ARGB444(3MLIB), mlib_VideoColorJFIFYCC2RGB444(3MLIB), attributes(5)`
The `mlib_VideoColorJFIFYCCK2CMYK444()` function performs color space conversion from YCbCrK to CMYK when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
\begin{align*}
R &= Y + 1.40200 \times (Cr - 128) \\
G &= Y - 0.34414 \times (Cb - 128) - 0.71414 \times (Cr - 128) \\
B &= Y + 1.77200 \times (Cb - 128) \\
C &= (255 - R) \\
M &= (255 - G) \\
Y &= (255 - B) \\
K &= K
\end{align*}
\]

The function takes the following arguments:

- `cmyk` Pointer to destination CMYK multi-component row. `cmyk` must be 8-byte aligned.
- `y` Pointer to source Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to source Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to source Cr component row. `cr` must be 8-byte aligned.
- `k` Pointer to source K component row. `k` must be 8-byte aligned.
- `n` Length of Y, Cb, Cr, and K component rows. The length of the CMYK multi-component row must be 4*n.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorCMYK2JFIFYCCK444(3MLIB), attributes(5)`
mlib_VideoColorMerge2 — color conversion (color channel merge)

Synopsis

```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorMerge2(mlib_u8 *colors,
       const mlib_u8 *color1,const mlib_u8 *color2,
       mlib_s32 n);
```

Description

The `mlib_VideoColorMerge2()` function performs color channel merge.

Parameters

The function takes the following arguments:

- **colors**: Pointer to colors multi-component row. Colors must be 8-byte aligned.
- **color1**: Pointer to first color component row. Color1 must be 8-byte aligned.
- **color2**: Pointer to second color component row. Color2 must be 8-byte aligned.
- **n**: Length of color1 and color2 arrays. Length of colors must be 2*n.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoColorMerge2_S16(3MLIB)`
- `mlib_VideoColorMerge3(3MLIB)`
- `mlib_VideoColorMerge3_S16(3MLIB)`
- `mlib_VideoColorMerge4(3MLIB)`
- `mlib_VideoColorMerge4_S16(3MLIB)`
- `mlib_VideoColorSplit2_S16(3MLIB)`
- `mlib_VideoColorSplit3(3MLIB)`
- `mlib_VideoColorSplit3_S16(3MLIB)`
- `mlib_VideoColorSplit4(3MLIB)`
- `mlib_VideoColorSplit4_S16(3MLIB)`
- attributes(5)
# mlib_VideoColorMerge2_S16

## Name
mlib_VideoColorMerge2_S16 - color conversion (color channel merge)

## Synopsis
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorMerge2_S16(mlib_s16 *colors,
const mlib_s16 *color1, const mlib_s16 *color2,
mlib_s32 n);
```

## Description
The `mlib_VideoColorMerge2_S16()` function performs color channel merge.

## Parameters
The function takes the following arguments:
- **colors**: Pointer to colors multi-component row. Colors must be 8-byte aligned.
- **color1**: Pointer to first color component row. Color1 must be 8-byte aligned.
- **color2**: Pointer to second color component row. Color2 must be 8-byte aligned.
- **n**: Length of color1 and color2 arrays. Length of colors must be 2*n.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
mlib_VideoColorMerge2(3MLIB), mlib_VideoColorMerge3(3MLIB),
mlib_VideoColorMerge3_S16(3MLIB), mlib_VideoColorMerge4(3MLIB),
mlib_VideoColorMerge4_S16(3MLIB), mlib_VideoColorSplit2(3MLIB),
mlib_VideoColorSplit2_S16(3MLIB), mlib_VideoColorSplit3(3MLIB),
mlib_VideoColorSplit3_S16(3MLIB), mlib_VideoColorSplit4(3MLIB),
mlib_VideoColorSplit4_S16(3MLIB), attributes(5)
The `mlib_VideoColorMerge3()` function performs color channel merge.

**Parameters**
The function takes the following arguments:

- `colors` Pointer to colors multi-component row. Colors must be 8-byte aligned.
- `color1` Pointer to first color component row. Color1 must be 8-byte aligned.
- `color2` Pointer to second color component row. Color2 must be 8-byte aligned.
- `color3` Pointer to third color component row. Color3 must be 8-byte aligned.
- `n` Length of color1, color2 and color3 arrays. Length of colors must be 3*n.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoColorMerge2(3MLIB), mlib_VideoColorMerge2_S16(3MLIB), mlib_VideoColorMerge3_S16(3MLIB), mlib_VideoColorMerge4(3MLIB), mlib_VideoColorMerge4_S16(3MLIB), mlib_VideoColorSplit2(3MLIB), mlib_VideoColorSplit2_S16(3MLIB), mlib_VideoColorSplit3(3MLIB), mlib_VideoColorSplit3_S16(3MLIB), mlib_VideoColorSplit4(3MLIB), mlib_VideoColorSplit4_S16(3MLIB), attributes(5)`
The `mlib_VideoColorMerge3_S16()` function performs color channel merge. The function takes the following arguments:

- **colors**: Pointer to colors multi-component row. Colors must be 8-byte aligned.
- **color1**: Pointer to first color component row. color1 must be 8-byte aligned.
- **color2**: Pointer to second color component row. color2 must be 8-byte aligned.
- **color3**: Pointer to third color component row. color3 must be 8-byte aligned.
- **n**: Length of color1, color2 and color3 arrays. Length of colors must be 3*n.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorMerge2(3MLIB)`, `mlib_VideoColorMerge2_S16(3MLIB)`, `mlib_VideoColorMerge3(3MLIB)`, `mlib_VideoColorMerge4(3MLIB)`, `mlib_VideoColorMerge4_S16(3MLIB)`, `mlib_VideoColorSplit2(3MLIB)`, `mlib_VideoColorSplit2_S16(3MLIB)`, `mlib_VideoColorSplit3(3MLIB)`, `mlib_VideoColorSplit3_S16(3MLIB)`, `mlib_VideoColorSplit4(3MLIB)`, `mlib_VideoColorSplit4_S16(3MLIB)`.

### Attributes

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See also `attributes(5)` for descriptions of the following attributes:
mlib_VideoColorMerge4 – color conversion (color channel merge)

Synopsis

```
c c ( flag... ) file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorMerge4(mlib_u8 *colors, const mlib_u8 *color1,
          const mlib_u8 *color2, const mlib_u8 *color3, const mlib_u8 *color4,
          mlib_s32 n);
```

Description
The mlib_VideoColorMerge4() function performs color channel merge.

Parameters
The function takes the following arguments:
- **colors** Pointer to colors multi-component row. colors must be 8-byte aligned.
- **color1** Pointer to first color component row. color1 must be 8-byte aligned.
- **color2** Pointer to second color component row. color2 must be 8-byte aligned.
- **color3** Pointer to third color component row. color3 must be 8-byte aligned.
- **color4** Pointer to fourth color component row. color4 must be 8-byte aligned.
- **n** Length of color1, color2, color3, and color4 arrays. Length of colors must be 4*n.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
- mlib_VideoColorMerge2(3MLIB), mlib_VideoColorMerge2_S16(3MLIB),
- mlib_VideoColorMerge3(3MLIB), mlib_VideoColorMerge3_S16(3MLIB),
- mlib_VideoColorMerge4_S16(3MLIB), mlib_VideoColorSplit2(3MLIB),
- mlib_VideoColorSplit2_S16(3MLIB), mlib_VideoColorSplit3(3MLIB),
- mlib_VideoColorSplit3_S16(3MLIB), mlib_VideoColorSplit4(3MLIB),
- mlib_VideoColorSplit4_S16(3MLIB), attributes(5)
The `mlib_VideoColorMerge4_S16()` function performs color channel merge.

### Parameters

The function takes the following arguments:

- **colors**: Pointer to colors multi-component row. Colors must be 8-byte aligned.
- **color1**: Pointer to first color component row. Color1 must be 8-byte aligned.
- **color2**: Pointer to second color component row. Color2 must be 8-byte aligned.
- **color3**: Pointer to third color component row. Color3 must be 8-byte aligned.
- **color4**: Pointer to fourth color component row. Color4 must be 8-byte aligned.
- **n**: Length of color1, color2, color3, and color4 arrays. Length of colors must be 4*n.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_VideoColorMerge2(3MLIB)`, `mlib_VideoColorMerge2_S16(3MLIB)`, `mlib_VideoColorMerge3(3MLIB)`, `mlib_VideoColorMerge3_S16(3MLIB)`, `mlib_VideoColorMerge4(3MLIB)`, `mlib_VideoColorSplit2(3MLIB)`, `mlib_VideoColorSplit2_S16(3MLIB)`, `mlib_VideoColorSplit3(3MLIB)`, `mlib_VideoColorSplit3_S16(3MLIB)`, `mlib_VideoColorSplit4(3MLIB)`, `mlib_VideoColorSplit4_S16(3MLIB)`, attributes(5)
The `mlib_VideoColorResizeABGR()` function resizes the source image with dimensions `src_w`, `src_h` into the destination image with dimensions `dst_w`, `dst_h` using nearest-neighbor, bilinear interpolation, or bicubic interpolation. The source buffer can contain multi-banded pixel stream, in which case, each band is resized independently. Edge conditions are handled according to the MLIB_EDGE_SRC_EXTEND scheme.

### Parameters

The function takes the following arguments:

- **dst**: Pointer to output image.
- **src**: Pointer to input image.
- **dst_w**: Output image width in pixels.
- **dst_h**: Output image height in rows.
- **dst_lb**: Input image width in pixels.
- **src_w**: Linebytes for input buffer.
- **src_h**: Input image height in lines.
- **src_lb**: Linebytes for input image.
- **filter**: Type of interpolation filter. It can be one of the following:
  - MLIB_NEAREST
  - MLIB_BILINEAR
  - MLIB_BICUBIC

### Return Values

None.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  attributes(5)
**mlib_VideoColorRGB2ABGR** – color conversion

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorRGB2ABGR(mlib_u8 *abgr, const mlib_u8 *rgb,
                                    mlib_s32 n);
```

**Description**
The `mlib_VideoColorRGB2ABGR()` function performs RGB to ABGR color order conversion.

**Parameters**
The function takes the following arguments:
- `abgr` Pointer to ABGR row.
- `rgb` Pointer to RGB row.
- `n` Number of pixels.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoColorABGR2RGB(3MLIB)`, `mlib_VideoColorARGB2RGB(3MLIB)`, `mlib_VideoColorRGB2ARGB(3MLIB)`, attributes(5)
The `mlib_VideoColorRGB2ARGB()` function performs RGB to ARGB color order conversion.

**Parameters**
- `argb` Pointer to ARGB row.
- `rgb` Pointer to RGB row.
- `n` Number of pixels.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoColorABGR2RGB(3MLIB)`, `mlib_VideoColorARGB2RGB(3MLIB)`, `mlib_VideoColorRGB2ABGR(3MLIB)`, attributes(5)
Name  mlib_VideoColorRGB2JFIFYCC420 – RGB to JFIF YCbCr color conversion

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorRGB2JFIFYCC420(mlib_u8 *y0, mlib_u8 *y1,
    mlib_u8 *cb, mlib_u8 *cr, const mlib_u8 *rgb0, const mlib_u8 *rgb1,
    mlib_s32 n);

Description  The mlib_VideoColorRGB2JFIFYCC420() function performs color space conversion from
RGB to YCbCr together with sampling rate conversion when used in the JPEG File
Interchange Format (JFIF).

Parameters  The function takes the following arguments:

  y0  Pointer to upper destination Y component row. y0 must be 8-byte aligned.

  y1  Pointer to lower destination Y component row. y1 must be 8-byte aligned.

  cb  Pointer to destination Cb component row. cb must be 8-byte aligned.

  cr  Pointer to destination Cr component row. cr must be 8-byte aligned.

  rgb0 Pointer to upper source RGB multi-component row. rgb0 must be 8-byte aligned.

  rgb1 Pointer to lower source RGB multi-component row. rgb1 must be 8-byte aligned.

  n   Length of Y component row. n must be even. The length of Cb and Cr component
      rows must be n/2. The length of the RGB multi-component row must be 3*n

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorABGR2JFIFYCC420(3MLIB), mlib_VideoColorABGR2JFIFYCC422(3MLIB),
mlib_VideoColorARGB2JFIFYCC420(3MLIB), mlib_VideoColorARGB2JFIFYCC422(3MLIB), attributes(5)
Name mlib_VideoColorRGB2JFIFYCC422 – RGB to JFIF YCbCr color conversion

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorRGB2JFIFYCC422(mlib_u8 *y, mlib_u8 *cb,
                                       mlib_u8 *cr, const mlib_u8 *rgb,
                                       mlib_s32 n);

Description The mlib_VideoColorRGB2JFIFYCC422() function performs color space conversion from RGB to YCbCr together with sampling rate conversion when used in the JPEG File Interchange Format (JFIF).

Parameters The function takes the following arguments:

- **y**: Pointer to destination Y component row. y must be 8-byte aligned.
- **cb**: Pointer to destination Cb component row. cb must be 8-byte aligned.
- **cr**: Pointer to destination Cr component row. cr must be 8-byte aligned.
- **rgb**: Pointer to source RGB multi-component row. rgb must be 8-byte aligned.
- **n**: Length of Y component row. n must be even. The length of Cb and Cr component rows must be n/2. The length of the RGB multi-component row must be 3*n.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VideoColorABGR2JFIFYCC420(3MLIB), mlib_VideoColorABGR2JFIFYCC422(3MLIB), mlib_VideoColorARGB2JFIFYCC420(3MLIB), mlib_VideoColorARGB2JFIFYCC422(3MLIB), mlib_VideoColorRGB2JFIFYCC420(3MLIB), attributes(5)
The `mlib_VideoColorRGB2JFIFYCC444()` function performs color space conversion from RGB to YCbCr when used in the JPEG File Interchange Format (JFIF).

The following equation is used:

\[
Y = 0.2990 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \\
Cb = -0.16874 \cdot R - 0.33126 \cdot G + 0.50000 \cdot B + 128 \\
Cr = 0.5000 \cdot R - 0.41869 \cdot G - 0.08131 \cdot B + 128
\]

The function takes the following arguments:

- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `rgb` Pointer to source RGB multi-component row. `rgb` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the RGB multi-component row must be 3*`n`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorABGR2JFIFYCC444(3MLIB)`, `mlib_VideoColorARGB2JFIFYCC444(3MLIB)`, `mlib_VideoColorRGB2JFIFYCC444_S16(3MLIB)`, `attributes(5)`
**Name**  
mlib_VideoColorRGB2JFIFYCC444_S16 - RGB to JFIF YCbCr color conversion

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
mlib_status mlib_VideoColorRGB2JFIFYCC444_S16(mlib_s16 *y, mlib_s16 *cb,
    mlib_s16 *cr, const mlib_s16 *rgb, mlib_s32 n);
```

**Description**  
The `mlib_VideoColorRGB2JFIFYCC444_S16()` function performs color space conversion from RGB to YCbCr when used in the JPEG File Interchange Format (JFIF).

Both the input RGB components and the output YCbCr components are supposed to be in the range of [0, 4095].

The following equation is used:

- Y = 0.29900 * R + 0.58700 * G + 0.11400 * B
- Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B + 2048
- Cr = 0.50000 * R - 0.41869 * G - 0.08131 * B + 2048

**Parameters**  
The function takes the following arguments:

- `y` Pointer to destination Y component row. `y` must be 8-byte aligned.
- `cb` Pointer to destination Cb component row. `cb` must be 8-byte aligned.
- `cr` Pointer to destination Cr component row. `cr` must be 8-byte aligned.
- `rgb` Pointer to source RGB multi-component row. `rgb` must be 8-byte aligned.
- `n` Length of Y component row. The length of Cb and Cr component rows must be `n`. The length of the RGB multi-component row must be `3*n`.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VideoColorABGR2JFIFYCC444(3MLIB), mlib_VideoColorARGB2JFIFYCC444(3MLIB), mlib_VideoColorRGB2JFIFYCC444(3MLIB), attributes(5)
mlib_VideoColorRGBAint_to_ABGRint

Name  mlib_VideoColorRGBAint_to_ABGRint – convert RGBA interleaved to ABGR

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
    #include <mlib.h>

    void mlib_VideoColorRGBAint_to_ABGRint(mlib_u32 *ABGR,
        const mlib_u32 *RGBA, mlib_s32 w,
        mlib_s32 h, mlib_s32 dlb,
        mlib_s32 slb);

Description  The RGBA pixel stream is broken apart and recombined into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The buffers have dimensions \(w\) and \(h\). Within each 32-bit input word, the component ordering is \(R\) (bits 31-24), \(G\) (bits 23-16), \(B\) (bits 15-8), and \(A\) (bits 7-0). Within each 32-bit output word, the component ordering is \(A\) (bits 31-24), \(B\) (bits 23-16), \(G\) (bits 15-8), and \(R\) (bits 7-0).

Parameters  The function takes the following arguments:

- \(ABGR\) Pointer to output buffer.
- \(RGBA\) Pointer to input buffer.
- \(w\) Image width in pixels.
- \(h\) Image height in lines.
- \(dlb\) Linebytes for output buffer.
- \(slb\) Linebytes for input buffer.

Return Values  None.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorABGRint_to_ARGBint(3MLIB),
          mlib_VideoColorBGRAint_to_ABGRint(3MLIB), attributes(5)
mlib_VideoColorRGBint_to_ABGRint(3MLIB)

Name
mlib_VideoColorRGBint_to_ABGRint – convert RGB interleaved to ABGR interleaved

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorRGBint_to_ABGRint(mlib_u32 *ABGR,
    const mlib_u8 *RGB, const mlib_u8 *A_array,
    mlib_u8 A_const, mlib_s32 w,
    mlib_s32 h, mlib_s32 dlб,
    mlib_s32 slб, mlib_s32 alb);

Description
The interleaved RGB stream, and the A values are combined into an A, B, G, R interleaved byte stream. Within each 24-bit input pixel, the component ordering is R (bits 23-16), G (bits 15-8), and B (bits 7-0). Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

The alpha values for this function work in the following fashion:
- If A_array pointer is not NULL, the values are taken from there. It has to have the same dimensions as the R, G, and B buffers.
- If A_array pointer is NULL, the alpha values for every pixel are set to A_const.

Parameters
The function takes the following arguments:

- **ABGR** Pointer to output buffer.
- **RGB** Pointer to input buffer.
- **A_array** Array of alpha values.
- **A_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlб** Line bytes for output buffer.
- **slб** Line bytes for input buffer.
- **alб** Line bytes for alpha buffer.

Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
mlib_VideoColorRGBseq_to_ABGRint

mlib_VideoColorBGRint_to_ABGRint

mlib_VideoColorRGBXint_to_ABGRint

mlib_VideoColorRGBXint_to_ARGBint

mlib_VideoColorXRGBint_to_ABGRint

mlib_VideoColorXRGBint_to_ARGBint

See Also  mlib_VideoColorRGBseq_to_ABGRint(3MLIB),
mlib_VideoColorBGRint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ARGBint(3MLIB),
mlib_VideoColorXRGBint_to_ABGRint(3MLIB),
mlib_VideoColorXRGBint_to_ARGBint(3MLIB), attributes(5)
mlib_VideoColorRGBint_to_BGRAint – convert RGB interleaved to BGRA interleaved

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorRGBint_to_BGRAint(mlib_u8 *bgra,
    const mlib_u8 *rgb,
    const mlib_u8 *a_array,
    mlib_u8 a_const,
    mlib_s32 w,
    mlib_s32 h,
    mlib_s32 dlb,
    mlib_s32 slb,
    mlib_s32 alb);

Description

The interleaved RGB stream and the A values are combined into an interleaved BGRA byte stream.

The alpha values for this function work in the following fashion:

■ If the a_array pointer is not NULL, the values are taken from there. It has to have the at least 1/3 the dimension of the RGB buffer.
■ If the a_array pointer is NULL, the alpha values for every pixel are set to a_const.

In other words, this function’s inner loop works like this:

bgra[0] = rgb[2];
bgra[1] = rgb[1];
bgra[2] = rgb[0];
bgra[3] = (a_array == NULL) ? a_const : a_array[0];

Parameters

The function takes the following arguments:

bgra Pointer to the output BGRA buffer.
rgb Pointer to the input RGB buffer.
a_array Pointer to the alpha buffer.
a_const Constant alpha value.
w Image width in pixels.
h Image height in lines.
dlb Line bytes of the output buffer.
slb Line bytes of the input buffer.
alb Line bytes of the alpha buffer.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:
mlib_VideoColorRGBint_to_BGRAint(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT·Level</td>
<td>MT·Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorRGBint_to_BGRAint(3MLIB), attributes(5)
Name: mlib_VideoColorRGBseq_to_ABGRint – convert RGB sequential to ABGR interleaved

Synopsis:
```c
#include <mlib.h>

void mlib_VideoColorRGBseq_to_ABGRint(mlib_u32 *ABGR, const mlib_u8 *R, const mlib_u8 *G, const mlib_u8 *B, const mlib_u8 *A_array, mlib_u8 A_const, mlib_s32 w, mlib_s32 h, mlib_s32 dlб, mlib_s32 slб);
```

Description:
The R, G, and B streams, and the A values are combined into an A, B, G, R interleaved byte stream. Within each 32-bit output word, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0).

The alpha values for this function work in the following fashion:
- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the R, G, and B buffers.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

Parameters:
The function takes the following arguments:
- `ABGR` Pointer to output buffer.
- `R` Pointer to input R buffer.
- `G` Pointer to input G buffer.
- `B` Pointer to input B buffer.
- `A_array` Array of alpha values.
- `A_const` Constant alpha value.
- `w` Image width in pixels.
- `h` Image height in lines.
- `dlб` Line bytes for output buffer.
- `slб` Line bytes for input buffers.

Return Values:
None.

Attributes:
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

mlib_VideoColorRGBseq_to_ABGRint(3MLIB),  
mlib_VideoColorBGRint_to_ABGRint(3MLIB),  
mlib_VideoColorRGBXint_to_ABGRint(3MLIB),  
mlib_VideoColorRGBXint_to_ARGBint(3MLIB),  
mlib_VideoColorXRGBint_to_ABGRint(3MLIB),  
mlib_VideoColorXRGBint_to_ARGBint(3MLIB), attributes(5)
The interleaved RGBX stream and the alpha values are combined into an interleaved A, B, G, R output stream. Within each 32-bit input pixel, the component ordering is R (bits 31-24), G (bits 23-16), and B (bits 15-8). Within each 32-bit output pixel, the component ordering is A (bits 31-24), B (bits 23-16), G (bits 15-8), and R (bits 7-0). The alpha values for this function work in the following fashion: if A_array is not NULL, the values are taken from the corresponding locations in the alpha array, otherwise a constant alpha value, specified by A_const, is stored in each output pixel. Each element in the alpha array is an unsigned byte. w and h define the dimensions of the region of the buffers to be processed. The linebyte parameters are used to advance the data pointers for each of the buffers.

Parameters

- **ABGR**  Pointer to output buffer (word-aligned).
- **RGBX**  Pointer to input buffer (word-aligned).
- **A_array**  Pointer to array of alpha values (byte-aligned).
- **A_const**  Constant alpha value (range = 0..255).
- **w**  Image width in pixels.
- **h**  Image height in lines.
- **dlb**  Linebytes for output buffer.
- **slb**  Linebytes for input buffer.
- **alb**  Linebytes for alpha buffer.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also
mlib_VideoColorRGBseq_to_ABGRint(3MLIB),
mlib_VideoColorRGBint_to_ABGRint(3MLIB),
mlib_VideoColorBGRint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ARGBint(3MLIB),
mlib_VideoColorXRGBint_to_ABGRint(3MLIB),
mlib_VideoColorXRGBint_to_ARGBint(3MLIB), attributes(5)
mlib_VideoColorRGBXint_to_ARGBint -- convert RGBX interleaved to ARGB interleaved

#include <mlib.h>

void mlib_VideoColorRGBXint_to_ARGBint(mlib_u32 *ARGB, 
    const mlib_u32 *RGBX, const mlib_u8 *A_array, 
    mlib_u8 A_const, mlib_s32 w, 
    mlib_s32 h, mlib_s32 dlb, 
    mlib_s32 slb, mlib_u32 alb);

Description
Similar to mlib_VideoColorRGBXint_to_ABGRint() except that the output component ordering is: A (bits 31-24), R (bits 23-16), G (bits 15-8), and B (bits 7-0).

Parameters
The function takes the following arguments:

- **ARGB** Pointer to output buffer (word-aligned).
- **RGBX** Pointer to input buffer (word-aligned).
- **A_array** Pointer to array of alpha values (byte-aligned).
- **A_const** Constant alpha value (range = 0..255).
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **slb** Linebytes for input buffer.
- **alb** Linebytes for alpha buffer.

Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoColorRGBseq_to_ABGRint(3MLIB),
mlib_VideoColorRGBint_to_ABGRint(3MLIB),
mlib_VideoColorBGRint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ABGRint(3MLIB),
mlib_VideoColorXRGBint_to_ABGRint(3MLIB),
mlib_VideoColorXRGBint_to_ARGBint(3MLIB),
attributes(5)
**mlib_VideoColorSplit2** – color conversion (color channel split)

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorSplit2(mlib_u8 *color1, mlib_u8 *color2,
                    const mlib_u8 *colors, mlib_s32 n);
```

**Description**

The `mlib_VideoColorSplit2()` function performs color channel split.

The elements of the `colors` array are alternately copied into the `color1` array and `color2` array.

**Parameters**

The function takes the following arguments:

- `color1` Pointer to first color component row. `color1` must be 8-byte aligned.
- `color2` Pointer to second color component row. `color2` must be 8-byte aligned.
- `colors` Pointer to colors multi-component row. `colors` must be 8-byte aligned.
- `n` Length of `color1` and `color2` arrays. Length of `colors` must be `2^n`.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_VideoColorMerge2(3MLIB)`, `mlib_VideoColorMerge2_S16(3MLIB)`,
`mlib_VideoColorMerge3(3MLIB)`, `mlib_VideoColorMerge3_S16(3MLIB)`,
`mlib_VideoColorMerge4(3MLIB)`, `mlib_VideoColorMerge4_S16(3MLIB)`,
`mlib_VideoColorSplit2_S16(3MLIB)`, `mlib_VideoColorSplit3(3MLIB)`,
`mlib_VideoColorSplit3_S16(3MLIB)`, `mlib_VideoColorSplit4(3MLIB)`,
`mlib_VideoColorSplit4_S16(3MLIB)`, `attributes(5)`
The `mlib_VideoColorSplit2_S16()` function performs color channel split. The elements of the colors array are alternately copied into the color1 array and color2 array.

### Parameters
- **color1**  Pointer to first color component row. color1 must be 8-byte aligned.
- **color2**  Pointer to second color component row. color2 must be 8-byte aligned.
- **colors**  Pointer to colors multi-component row. colors must be 8-byte aligned.
- **n**  Length of color1 and color2 arrays. Length of colors must be 2^n.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_VideoColorMerge2(3MLIB)`, `mlib_VideoColorMerge2_S16(3MLIB)`,
- `mlib_VideoColorMerge3(3MLIB)`, `mlib_VideoColorMerge3_S16(3MLIB)`,
- `mlib_VideoColorMerge4(3MLIB)`, `mlib_VideoColorMerge4_S16(3MLIB)`,
- `mlib_VideoColorSplit2(3MLIB)`, `mlib_VideoColorSplit3(3MLIB)`,
The `mlib_VideoColorSplit3()` function performs color channel split. The elements of the colors array are selected in consecutive groups of three. As each group is processed, the first element is stored in the color1 array; the second element, in the color2 array; and the third element, in the color3 array. This process is repeated until the end of the colors array is reached.

The function takes the following arguments:

- **color1** Pointer to first color component row. color1 must be 8-byte aligned.
- **color2** Pointer to second color component row. color2 must be 8-byte aligned.
- **colors** Pointer to colors multi-component row. colors must be 8-byte aligned.
- **n** Length of color1, color2, and color3 arrays. Length of colors must be 3*n.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorSplit2(3MLIB)`, `mlib_VideoColorSplit2_S16(3MLIB)`, `mlib_VideoColorSplit3(3MLIB)`, `mlib_VideoColorSplit3_S16(3MLIB)`, `mlib_VideoColorSplit4(3MLIB)`
The `mlib_VideoColorSplit3_S16()` function performs color channel split.

The elements of the colors array are selected in consecutive groups of three. As each group is processed, the first element is stored in the `color1` array; the second element, in the `color2` array; and the third element, in the `color3` array. This process is repeated until the end of the colors array is reached.

**Parameters**

The function takes the following arguments:

- `color1` Pointer to first color component row. `color1` must be 8-byte aligned.
- `color2` Pointer to second color component row. `color2` must be 8-byte aligned.
- `colors` Pointer to colors multi-component row. `colors` must be 8-byte aligned.
- `n` Length of `color1`, `color2`, and `color3` arrays. Length of `colors` must be 3*n.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoColorMerge2(3MLIB)`, `mlib_VideoColorMerge2_S16(3MLIB)`
- `mlib_VideoColorMerge3(3MLIB)`, `mlib_VideoColorMerge3_S16(3MLIB)`
- `mlib_VideoColorMerge4(3MLIB)`, `mlib_VideoColorMerge4_S16(3MLIB)`
- `mlib_VideoColorSplit2(3MLIB)`, `mlib_VideoColorSplit2_S16(3MLIB)`
- `mlib_VideoColorSplit3(3MLIB)`, `mlib_VideoColorSplit4(3MLIB)`
- `mlib_VideoColorSplit4_S16(3MLIB)`, attributes(5)
mlib_VideoColorSplit4 – color conversion (color channel split)

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorSplit4(mlib_u8 *color1, mlib_u8 *color2,
       mlib_u8 *color3, mlib_u8 *color4, const mlib_u8 *colors, mlib_s32 n);

Description

The mlib_VideoColorSplit4() function performs color channel split.

The elements of the colors array are selected in consecutive groups of four. As each group is processed, the first element is stored in the color1 array; the second element, in the color2 array; and so on. This process is repeated until the end of the colors array is reached.

Parameters

The function takes the following arguments:

color1 Pointer to first color component row. color1 must be 8-byte aligned.
color2 Pointer to second color component row. color2 must be 8-byte aligned.
color3 Pointer to third color component row. color3 must be 8-byte aligned.
color4 Pointer to fourth color component row. color4 must be 8-byte aligned.
colors Pointer to colors multi-component row. colors must be 8-byte aligned.
n Length of color1, color2, color3, and color4 arrays. Length of colors must be 4*n.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_VideoColorMerge2(3MLIB), mlib_VideoColorMerge2_S16(3MLIB),
mlib_VideoColorMerge3(3MLIB), mlib_VideoColorMerge3_S16(3MLIB),
mlib_VideoColorMerge4(3MLIB), mlib_VideoColorMerge4_S16(3MLIB),
mlib_VideoColorSplit2(3MLIB), mlib_VideoColorSplit2_S16(3MLIB),
mlib_VideoColorSplit3(3MLIB), mlib_VideoColorSplit3_S16(3MLIB),
mlib_VideoColorSplit4_S16(3MLIB), attributes(5)
Name  
mlib_VideoColorSplit4_S16 – color conversion (color channel split)

Synopsis  
cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorSplit4_S16(mlib_s16 *color1,
    mlib_s16 *color2, mlib_s16 *color3,
    mlib_s16 *color4, const mlib_s16 *colors,
    mlib_s32 n);

Description  
The mlib_VideoColorSplit4_S16() function performs color channel split.

The elements of the colors array are selected in consecutive groups of four. As each group is
processed, the first element is stored in the color1 array; the second element, in the color2
array; and so on. This process is repeated until the end of the colors array is reached.

Parameters  
The function takes the following arguments:

- color1  Pointer to first color component row. color1 must be 8-byte aligned.
- color2  Pointer to second color component row. color2 must be 8-byte aligned.
- colors  Pointer to colors multi-component row. colors must be 8-byte aligned.
- n       Length of color1, color2, color3, and color4 arrays. Length of colors must be 4*n.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoColorSplit2_S16(3MLIB), mlib_VideoColorSplit2_S16(3MLIB),
mlib_VideoColorSplit3_S16(3MLIB), mlib_VideoColorSplit3_S16(3MLIB),
mlib_VideoColorSplit4_S16(3MLIB), mlib_VideoColorSplit4_S16(3MLIB),
mlib_VideoColorSplit2_S16(3MLIB), mlib_VideoColorSplit2_S16(3MLIB),
mlib_VideoColorSplit3_S16(3MLIB), mlib_VideoColorSplit3_S16(3MLIB),
mlib_VideoColorSplit4_S16(3MLIB), attributes(5)
The UYV pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions \( w \) and \( h \).

The alpha values for this function work in the following fashion:

- If \( A \_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \( Y \) buffer.
- If \( A \_array \) pointer is NULL, the alpha values for every pixel are set to \( A \_const \).

The following equation is used:

\[
\begin{align*}
[R] &= \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \end{bmatrix} \begin{bmatrix} [Y] & 16.0000 \end{bmatrix} \\
[G] &= \begin{bmatrix} 1.1644 & -0.3920 & -0.8132 \end{bmatrix} * \begin{bmatrix} [U] - 128.0000 \end{bmatrix} \\
[B] &= \begin{bmatrix} 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} [V] & 128.0000 \end{bmatrix}
\end{align*}
\]

Parameters
The function takes the following arguments:

- \( ABGR \): Pointer to output buffer.
- \( UYV \): Pointer to input buffer.
- \( A \_array \): Array of alpha values.
- \( A \_const \): Constant alpha value.
- \( w \): Image width in pixels.
- \( h \): Image height in lines.
- \( dlb \): Linebytes for output buffer.
- \( slb \): Linebytes for input buffer.
- \( alb \): Linebytes for alpha buffer.

Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:
mlib_VideoColorYUV444int_to_ABGRint(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUYV444int_to_ABGRint(3MLIB),
mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUV444int_to_ARGBint(3MLIB), attributes(5)
mlib_VideoColorUYV444int_to_ARGBint – color convert UYV interleaved to ARGB interleaved

The UYV pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions \( w \) and \( h \).

The alpha values for this function work in the following fashion:

- If \( A_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \( Y \) buffer.
- If \( A_array \) pointer is NULL, the alpha values for every pixel are set to \( A_{\text{const}} \).

The following equation is used:

\[
\begin{align*}
|R| & = 1.1644 \begin{bmatrix} 1.0000 & 0.5966 \end{bmatrix} \begin{bmatrix} Y \end{bmatrix} + 16.0000
\end{align*}
\]
\[
\begin{align*}
|G| & = 1.1644 \begin{bmatrix} -0.3920 & -0.8132 \end{bmatrix} \begin{bmatrix} U \end{bmatrix} - 128.0000
\end{align*}
\]
\[
\begin{align*}
|B| & = 1.1644 \begin{bmatrix} 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} V \end{bmatrix} + 128.0000
\end{align*}
\]

Parameters

The function takes the following arguments:

- **ARGB** Pointer to output buffer.
- **UYV** Pointer to input buffer.
- **A_array** Array of alpha values.
- **A_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **slb** Linebytes for input buffer.
- **alb** Linebytes for alpha buffer.

Return Values

None.

Attributes

See [attributes(5)] for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),  
mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),  
mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),  
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),  
mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),  
mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),  
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),  
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),  
mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),  
mlib_VideoColorYUYV444int_to_ARGBint(3MLIB),  
mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),  
mlib_VideoColorUYY422int_to_ARGBint(3MLIB),  
mlib_VideoColorUYY444int_to_ARGBint(3MLIB),  
mlib_VideoColorUYY422int_to_ABGRint(3MLIB),  
mlib_VideoColorUYV444int_to_ABGRint(3MLIB), attributes(5)
mlib_VideoColorUYV444int_to_UYVY422int – convert UYV interleaved with subsampling

Synopsis

```c
void mlib_VideoColorUYV444int_to_UYVY422int(mlib_u32 *UYVY, const mlib_u8 *UYV, mlib_s32 w, mlib_s32 h, mlib_s32 dlb, mlib_s32 slb);
```

Description

The UYV pixel stream is broken apart and recombined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The UYV buffer has dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is \( U[r][c], Y[r][c], V[r][c], U[r][c+1], Y[r][c+1], V[r][c+1], \ldots \)

The following equation is used:

\[
UYVY[r][c/2] = \left( \frac{(U[r][c] + U[r][c+1])}{2} \right) \ll 24 | \left( \frac{Y[r][c]}{2} \right) \ll 16 | \left( \frac{(V[r][c] + V[r][c+1])}{2} \right) \ll 8 | \left( \frac{Y[r][c+1]}{2} \right)
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

Parameters

- **UYVY**: Pointer to output buffer.
- **UYV**: Pointer to input buffer.
- **w**: Image width in pixels.
- **h**: Image height in lines.
- **dlb**: Linebytes for output buffer.
- **slb**: Linebytes for input buffer.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- mlib_VideoColorYUV444seq_to_YUYV422int(3MLIB)
- mlib_VideoColorYUV444int_to_YUYV422int(3MLIB)
- mlib_VideoColorYUV444seq_to_UYVY422int(3MLIB)
mlib_VideoColorYUV444int_to_UYVY422int(3MLIB),
mlib_VideoColorUYV444int_to_YUYV422int(3MLIB), attributes(5)
The UYV pixel stream is broken apart and recombined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The UYV buffer has dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is \( U[r][c], Y[r][c], V[r][c], U[r][c+1], Y[r][c+1], V[r][c+1], \ldots \)

The following equation is used:

\[
YUYV[r][c/2] = (Y[r][c] \ll 24) | \\
((U[r][c] + U[r][c+1]) / 2) \ll 16) | \\
(Y[r][c+1] \ll 8) | \\
((V[r][c] + V[r][c+1]) / 2))
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

The function takes the following arguments:

- **YUYV** Pointer to output buffer.
- **UYV** Pointer to input buffer.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **slb** Linebytes for input buffer.

### Return Values
None.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- mlib_VideoColorYUV444seq_to_YUYV422int(3MLIB),
- mlib_VideoColorYUV444int_to_YUYV422int(3MLIB),
- mlib_VideoColorYUV444seq_to_UYVY422int(3MLIB),
mlib_VideoColorYUV444int_to_UYVY422int(3MLIB)

mlib_VideoColorUYV444int_to_UYVY422int(3MLIB),
mlib_VideoColorUYV444int_to_UYVY422int(3MLIB), attributes(5)
mlib_VideoColorUYVY422int_to_ABGRint–color convert UYVY interleaved to ABGR interleaved

**Synopsis**

```c
#include <mlib.h>

void mlib_VideoColorUYVY422int_to_ABGRint(mlib_u32 *ABGR, 
    const mlib_u32 *UYVY, const mlib_u8 *A_array, 
    mlib_u8 A_const, mlib_s32 w, mlib_s32 h, mlib_s32 dlb, 
    mlib_s32 slb, mlib_s32 alb);
```

**Description**

The UYVY pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The UYVY buffer has dimensions \( \frac{w}{2} \) and \( h \). The ABGR buffer has dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be even.

The alpha values for this function work in the following fashion:

- If \( A \_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \( Y \) buffer.
- If \( A \_array \) pointer is NULL, the alpha values for every pixel are set to \( A \_const \).

The following equation is used:

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \\ 1.1644 & -0.3920 & -0.8132 \\ 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} - \begin{bmatrix} 16.0000 \\ 128.0000 \\ 128.0000 \end{bmatrix}
\]

**Parameters**

The function takes the following arguments:

- **ABGR** Pointer to output buffer.
- **UYVY** Pointer to input buffer.
- **A_array** Array of alpha values.
- **A_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **slb** Linebytes for input buffer.
- **alb** Linebytes for alpha buffer.

**Return Values**

None.

**Attributes**

See attributes(5) for descriptions of the following attributes:
mlib_VideoColorUYVY422int_to_ABGRint(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),  
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),  
mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),  
mlib_VideoColorYUYV444int_to_ABGRint(3MLIB),  
mlib_VideoColorUYVY422int_to_ARGBint(3MLIB),  
mlib_VideoColorUYVY444int_to_ARGBint(3MLIB),  
mlib_VideoColorUYVY422int_to_ABGRint(3MLIB),  
mlib_VideoColorUYVY444int_to_ABGRint(3MLIB),  
mlib_VideoColorUYVY444int_to_ABGRint(3MLIB),  
mlib_VideoColorUYVY444int_to_ABGRint(3MLIB), attributes(5)
The UYVY pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The UYVY buffer has dimensions \( w/2 \) and \( h \). The ARGB buffer has dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be even.

The alpha values for this function work in the following fashion:

- If \( A\_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \( Y \) buffer.
- If \( A\_array \) pointer is NULL, the alpha values for every pixel are set to \( A\_const \).

The following equation is used:

\[
\begin{align*}
|R| &= [1.1644 \quad 0.0000 \quad 1.5966] \quad [|Y| \quad 16.0000] \\
|G| &= [1.1644 \quad -0.3920 \quad -0.8132] \quad [|U| \quad -128.0000] \\
|B| &= [1.1644 \quad 2.0184 \quad 0.0000] \quad [|V| \quad 128.0000]
\end{align*}
\]

The function takes the following arguments:

- \( ARGB \) Pointer to output buffer.
- \( UYVY \) Pointer to input buffer.
- \( A\_array \) Array of alpha values.
- \( A\_const \) Constant alpha value.
- \( w \) Image width in pixels.
- \( h \) Image height in lines.
- \( dlb \) Linebytes for output buffer.
- \( slb \) Linebytes for input buffer.
- \( alb \) Linebytes for alpha buffer.

None.

See attributes(5) for descriptions of the following attributes:
mlib_VideoColorUYVY422int_to_ARGBint(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
- mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
- mlib_VideoColorYUV444int_to_ARGBint(3MLIB),
- mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
- mlib_VideoColorYUV444int_to_ABGRint(3MLIB),
- mlib_VideoColorUYVY422int_to_ABGRint(3MLIB),
- mlib_VideoColorUYV444int_to_ARGBint(3MLIB),
- mlib_VideoColorUYV444int_to_ABGRint(3MLIB), attributes(5)
Name  mlib_VideoColorXRGBint_to_ABGRint – convert XRGB interleaved to ABGR interleaved

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorXRGBint_to_ABGRint(mlib_u32 *ABGR,
const mlib_u32 *XRGB, const mlib_u8 *A_array,
mlib_u8 A_const, mlib_s32 w, mlib_s32 h,
mlib_s32 dlb, mlib_s32 slb, mlib_s32 alb);

Description  Similar to mlib_VideoColorRGBXint_to_ABGRint except that the input component ordering is: R (bits 23-16), G (bits 15-8), and B (bits 7-0).

Parameters  The function takes the following arguments:

- **ABGR**: Pointer to output buffer (word-aligned).
- **XRGB**: Pointer to input buffer (word-aligned).
- **A_array**: Pointer to array of alpha values (byte-aligned).
- **A_const**: Constant alpha value (range = 0..255).
- **w**: Image width in pixels.
- **h**: Image height in lines.
- **dlb**: Linebytes for output buffer.
- **slb**: Linebytes for input buffer.
- **alb**: Linebytes for alphabuffer.

Return Values  None.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorRGBseq_to_ABGRint(3MLIB),
mlib_VideoColorRGBint_to_ABGRint(3MLIB),
mlib_VideoColorBGRint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ABGRint(3MLIB),
mlib_VideoColorRGBXint_to_ARGBint(3MLIB),
mlib_VideoColorXRGBint_to_ARGBint(3MLIB), attributes(5)
**Name**  
mlib_VideoColorXRGBint_to_ARGBint – convert XRGB interleaved to ARGB interleaved

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

```c
void mlib_VideoColorXRGBint_to_ARGBint(mlib_u32 *ARGB,
    const mlib_u32 *XRGB, const mlib_u8 *A_array,
    mlib_u8 A_const, mlib_s32 w, mlib_s32 h,
    mlib_s32 db, mlib_s32 sb, mlib_s32 alb);
```

**Description**  
Similar to mlib_VideoColorRGBXint_to_ARGBint except that the input component ordering is: R (bits 23-16), G (bits 15-8), and B (bits 7-0).

**Parameters**  
The function takes the following arguments:

- **ARGB**  
  Pointer to output buffer (word-aligned).

- **XRGB**  
  Pointer to input buffer (word-aligned).

- **A_array**  
  Pointer to array of alpha values (byte-aligned).

- **A_const**  
  Constant alpha value (range = 0..255).

- **w**  
  Image width in pixels.

- **h**  
  Image height in lines.

- **db**  
  Linebytes for output buffer.

- **sb**  
  Linebytes for input buffer.

- **alb**  
  Linebytes for alphabuffer.

**Return Values**  
None.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VideoColorRGBseq_to_ABGRint(3MLIB),  
mlib_VideoColorRGBint_to_ABGRint(3MLIB),  
mlib_VideoColorBGint_to_ABGRint(3MLIB),  
mlib_VideoColorRGBXint_to_ABGRint(3MLIB),  
mlib_VideoColorRGBXint_to_ARGBint(3MLIB),  
mlib_VideoColorXRGBint_to_ABGRint(3MLIB),  
mlib_VideoColorXRGBint_to_ARGBint(3MLIB), attributes(5)
mlib_VideoColorYUV2ABGR411

# include <mlib.h>

mlib_status mlib_VideoColorYUV2ABGR411(mlib_u8 *abgr, const mlib_u8 *y, 
const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height, 
mlib_s32 rgb_stride, mlib_s32 y_stride, mlib_s32 uv_stride);

Description
The mlib_VideoColorYUV2ABGR411() function performs YUV to RGB color conversion used 
in digital video compression in the 4:1:1 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and 
V, respectively. The size of the chrominance image depends on the chroma format used by the 
sequence. In this sequence, the chrominance components are subsampled 4-to-1 in only the 
horizontal direction in respect to the luminance component.

Parameters
The function takes the following arguments:

- abgr: Pointer to the destination packed ABGR image.
- y: Pointer to the source Y component.
- u: Pointer to the source U component.
- v: Pointer to the source V component.
- width: Width of the image.
- height: Height of the image.
- rgb_stride: Stride, in bytes, between adjacent rows in the destination image.
- y_stride: Stride, in bytes, between adjacent rows in the Y component image.
- uv_stride: Stride, in bytes, between adjacent rows in the U and V component images.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoColorYUV2ABGR420(3MLIB), mlib_VideoColorYUV2ABGR422(3MLIB), 
mlib_VideoColorYUV2ABGR444(3MLIB), mlib_VideoColorYUV2ARGB411(3MLIB), 
mlib_VideoColorYUV2ARGB420(3MLIB), mlib_VideoColorYUV2ARGB422(3MLIB), 
mlib_VideoColorYUV2ARGB444(3MLIB), mlib_VideoColorYUV2RGB411(3MLIB), 
mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB), 
mlib_VideoColorYUV2RGB444(3MLIB), mlib_VideoColorYUV2RGB411(3MLIB),
mlib_VideoColorYUV2ABGR411(3MLIB)

mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
The `mlib_VideoColorYUV2ABGR420()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

The function takes the following arguments:

- `abgr` Pointer to the destination packed ABGR image.
- `y` Pointer to the source Y component.
- `u` Pointer to the source U component.
- `v` Pointer to the source V component.
- `width` Width of the image.
- `height` Height of the image.
- `rgb_stride` Stride, in bytes, between adjacent rows in the destination image.
- `y_stride` Stride, in bytes, between adjacent rows in the Y component image.
- `uv_stride` Stride, in bytes, between adjacent rows in the U and V component images.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorYUV2ABGR411(3MLIB), mlib_VideoColorYUV2ABGR422(3MLIB), mlib_VideoColorYUV2ABGR444(3MLIB), mlib_VideoColorYUV2ARGB411(3MLIB), mlib_VideoColorYUV2ARGB420(3MLIB), mlib_VideoColorYUV2ARGB422(3MLIB), mlib_VideoColorYUV2ARGB444(3MLIB), mlib_VideoColorYUV2RGB411(3MLIB), mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB), mlib_VideoColorYUV2RGB444(3MLIB), mlib_VideoColorYUV2RGB555(3MLIB), mlib_VideoColorYUV2RGB888(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB), mlib_VideoColorYUV2RGB444(3MLIB), mlib_VideoColorYUV2RGB555(3MLIB), mlib_VideoColorYUV2RGB888(3MLIB), mlib_VideoColorYUV2RGB555(3MLIB), mlib_VideoColorYUV2RGB888(3MLIB).`
mlib_VideoColorYUV2ABGR420(3MLIB), mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB), mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
### Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_VideoColorYUV2ABGR420_W(mlib_u8 *abgr, const mlib_u8 *y, const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height, mlib_s32 abgr_stride, mlib_s32 y_stride, mlib_s32 uv_stride, mlib_s32 left, mlib_s32 top, mlib_s32 right, mlib_s32 bottom);
```

### Description

The `mlib_VideoColorYUV2ABGR420_W()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

### Parameters

- **abgr**: Pointer to the destination packed ABGR image. abgr must be 8-byte aligned.
- **y**: Pointer to the source Y component. y must be 8-byte aligned.
- **u**: Pointer to the source U component. u must be 4-byte aligned.
- **v**: Pointer to the source V component. v must be 4-byte aligned.
- **width**: Width of the image. width must be a multiple of 8.
- **height**: Height of the image. height must be a multiple of 2.
- **abgr_stride**: Stride, in bytes, between adjacent rows in the ABGR image. abgr_stride must be a multiple of 8.
- **y_stride**: Stride, in bytes, between adjacent rows in the Y component image. y_stride must be a multiple of 8.
- **uv_stride**: Stride, in bytes, between adjacent rows in the U and V component images. uv_stride must be a multiple of 8.
- **left**: Left border of clipping window. \( 0 \leq \text{left} < \text{right} \leq \text{width} \).
- **top**: Top border of clipping window. \( 0 \leq \text{top} < \text{bottom} \leq \text{height} \).
- **right**: Left border of clipping window. \( 0 \leq \text{left} < \text{right} \leq \text{width} \).
- **bottom**: Bottom border of clipping window. \( 0 \leq \text{top} < \text{bottom} \leq \text{height} \).
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorYUV2ABGR420_WX2(3MLIB), mlib_VideoColorYUV2ABGR420_WX3(3MLIB), mlib_VideoColorYUV2ABGR420_X2(3MLIB), mlib_VideoColorYUV2ABGR420_X3(3MLIB), attributes(5)
The `mlib_VideoColorYUV2ABGR420_WX2()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping and 2X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

The function takes the following arguments:

- `abgr`: Pointer to the destination packed ABGR image. `abgr` must be 8-byte aligned.
- `y`: Pointer to the source Y component. `y` must be 8-byte aligned.
- `u`: Pointer to the source U component. `u` must be 4-byte aligned.
- `v`: Pointer to the source V component. `v` must be 4-byte aligned.
- `width`: Width of the image. `width` must be a multiple of 8.
- `height`: Height of the image. `height` must be a multiple of 2.
- `abgr_stride`: Stride, in bytes, between adjacent rows in the ABGR image. `abgr_stride` must be a multiple of 8.
- `y_stride`: Stride, in bytes, between adjacent rows in the Y component image. `y_stride` must be a multiple of 8.
- `uv_stride`: Stride, in bytes, between adjacent rows in the U and V component images. `uv_stride` must be a multiple of 8.
- `left`: Left border of clipping window. `0 ≤ left < right ≤ width`.
- `top`: Top border of clipping window. `0 ≤ top < bottom ≤ height`.
- `right`: Left border of clipping window. `0 ≤ left < right ≤ width`.
- `bottom`: Bottom border of clipping window. `0 ≤ top < bottom ≤ height`.

### Parameters

- `abgr`: Pointer to the destination packed ABGR image. `abgr` must be 8-byte aligned.
- `y`: Pointer to the source Y component. `y` must be 8-byte aligned.
- `u`: Pointer to the source U component. `u` must be 4-byte aligned.
- `v`: Pointer to the source V component. `v` must be 4-byte aligned.
- `width`: Width of the image. `width` must be a multiple of 8.
- `height`: Height of the image. `height` must be a multiple of 2.
- `abgr_stride`: Stride, in bytes, between adjacent rows in the ABGR image. `abgr_stride` must be a multiple of 8.
- `y_stride`: Stride, in bytes, between adjacent rows in the Y component image. `y_stride` must be a multiple of 8.
- `uv_stride`: Stride, in bytes, between adjacent rows in the U and V component images. `uv_stride` must be a multiple of 8.
- `left`: Left border of clipping window. `0 ≤ left < right ≤ width`.
- `top`: Top border of clipping window. `0 ≤ top < bottom ≤ height`.
- `right`: Left border of clipping window. `0 ≤ left < right ≤ width`.
- `bottom`: Bottom border of clipping window. `0 ≤ top < bottom ≤ height`.

### Description

The `mlib_VideoColorYUV2ABGR420_WX2()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping and 2X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorYUV2ABGR420_W(3MLIB), mlib_VideoColorYUV2ABGR420_WX3(3MLIB), mlib_VideoColorYUV2ABGR420_X2(3MLIB), mlib_VideoColorYUV2ABGR420_X3(3MLIB), attributes(5)
# mlib_VideoColorYUV2ABGR420_WX3

The mlib_VideoColorYUV2ABGR420_WX3() function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with window clipping and 3X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

## Parameters

The function takes the following arguments:

- **abgr**: Pointer to the destination packed ABGR image. abgr must be 8-byte aligned.
- **y**: Pointer to the source Y component. y must be 8-byte aligned.
- **u**: Pointer to the source U component. u must be 4-byte aligned.
- **v**: Pointer to the source V component. v must be 4-byte aligned.
- **width**: Width of the image. width must be a multiple of 8.
- **height**: Height of the image. height must be a multiple of 2.
- **abgr_stride**: Stride, in bytes, between adjacent rows in the ABGR image. abgr_stride must be a multiple of 8.
- **y_stride**: Stride, in bytes, between adjacent rows in the Y component image. y_stride must be a multiple of 8.
- **uv_stride**: Stride, in bytes, between adjacent rows in the U and V component images. uv_stride must be a multiple of 8.
- **left**: Left border of clipping window. 0 ≤ left < right ≤ width.
- **top**: Top border of clipping window. 0 ≤ top < bottom ≤ height.
- **right**: Left border of clipping window. 0 ≤ left < right ≤ width.
- **bottom**: Bottom border of clipping window. 0 ≤ top < bottom ≤ height.
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlb_VideoColorYUV2ABGR420_W(3MLIB), mlb_VideoColorYUV2ABGR420_WX2(3MLIB), mlb_VideoColorYUV2ABGR420_X2(3MLIB), mlb_VideoColorYUV2ABGR420_X3(3MLIB), attributes(5)
The `mlib_VideoColorYUV2ABGR420_X2()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with 2X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

### Parameters

- **abgr**: Pointer to the destination packed ABGR image. abgr must be 8-byte aligned.
- **y**: Pointer to the source Y component. y must be 8-byte aligned.
- **u**: Pointer to the source U component. u must be 4-byte aligned.
- **v**: Pointer to the source V component. v must be 4-byte aligned.
- **width**: Width of the image. width must be a multiple of 8.
- **height**: Height of the image. height must be a multiple of 2.
- **abgr_stride**: Stride, in bytes, between adjacent rows in the ABGR image. abgr_stride must be a multiple of 8.
- **y_stride**: Stride, in bytes, between adjacent rows in the Y component image. y_stride must be a multiple of 8.
- **uv_stride**: Stride, in bytes, between adjacent rows in the U and V component images. uv_stride must be a multiple of 8.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VideoColorYUV2ABGR420_W(3MLIB), mlib_VideoColorYUV2ABGR420_WX2(3MLIB),
mlib_VideoColorYUV2ABGR420_WX3(3MLIB), mlib_VideoColorYUV2ABGR420_X3(3MLIB),
attributes(5)
### Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_VideoColorYUV2ABGR420_X3(mlib_u8 *abgr, const mlib_u8 *y,
const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height,
mlib_s32 abgr_stride, mlib_s32 y_stride, mlib_s32 uv_stride);
```

### Description

The `mlib_VideoColorYUV2ABGR420_X3()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence. It performs color conversion together with 3X zooming.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

### Parameters

- `abgr` Pointer to the destination packed ABGR image. `abgr` must be 8-byte aligned.
- `y` Pointer to the source Y component. `y` must be 8-byte aligned.
- `u` Pointer to the source U component. `u` must be 4-byte aligned.
- `v` Pointer to the source V component. `v` must be 4-byte aligned.
- `width` Width of the image. `width` must be a multiple of 8.
- `height` Height of the image. `height` must be a multiple of 2.
- `abgr_stride` Stride, in bytes, between adjacent rows in the ABGR image. `abgr_stride` must be a multiple of 8.
- `y_stride` Stride, in bytes, between adjacent rows in the Y component image. `y_stride` must be a multiple of 8.
- `uv_stride` Stride, in bytes, between adjacent rows in the U and V component images. `uv_stride` must be a multiple of 8.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also

mlib_VideoColorYUV2ABGR420_W(3MLIB), mlib_VideoColorYUV2ABGR420_WX2(3MLIB),
mlib_VideoColorYUV2ABGR420_WX3(3MLIB), mlib_VideoColorYUV2ABGR420_X2(3MLIB),
attributes(5)
**Name**
mlib_VideoColorYUV2ABGR422 – YUV to RGB color conversion

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2ABGR422(mlib_u8 *abgr, const mlib_u8 *y,
    const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height,
    mlib_s32 rgb_stride, mlib_s32 y_stride, mlib_s32 uv_stride);
```

**Description**
The `mlib_VideoColorYUV2ABGR422()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:2 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters**
The function takes the following arguments:
- `abgr` Pointer to the destination packed ABGR image.
- `y` Pointer to the source Y component.
- `u` Pointer to the source U component.
- `v` Pointer to the source V component.
- `width` Width of the image.
- `height` Height of the image.
- `rgb_stride` Stride, in bytes, between adjacent rows in the destination image.
- `y_stride` Stride, in bytes, between adjacent rows in the Y component image.
- `uv_stride` Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoColorYUV2ABGR411(3MLIB)`
- `mlib_VideoColorYUV2ABGR420(3MLIB)`
- `mlib_VideoColorYUV2ABGR444(3MLIB)`
- `mlib_VideoColorYUV2ARGB411(3MLIB)`
- `mlib_VideoColorYUV2ARGB420(3MLIB)`
- `mlib_VideoColorYUV2ARGB422(3MLIB)`
- `mlib_VideoColorYUV2ARGB444(3MLIB)`
- `mlib_VideoColorYUV2RGB411(3MLIB)`
mlib_VideoColorYUV2ABGR422(3MLIB), 
mllib_VideoColorYUV2RGB420(3MLIB), mllib_VideoColorYUV2RGB422(3MLIB), 
mllib_VideoColorYUV2RGB444(3MLIB), attributes(5)
mlib_VideoColorYUV2ABGR444 – YUV to RGB color conversion

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2ABGR444(mlib_u8 *abgr, const mlib_u8 *y,
                          const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height,
                          mlib_s32 rgb_stride, mlib_s32 yuv_stride);

Description  The mlib_VideoColorYUV2ABGR444() function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:4:4 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components have the same resolution as the luminance component.

Parameters  The function takes the following arguments:
- abgr  Pointer to the destination packed ABGR image.
- y  Pointer to the source Y component.
- u  Pointer to the source U component.
- v  Pointer to the source V component.
- width  Width of the image.
- height  Height of the image.
- rgb_stride  Stride, in bytes, between adjacent rows in the destination image.
- yuv_stride  Stride, in bytes, between adjacent rows in the source image.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorYUV2ABGR411(3MLIB), mlib_VideoColorYUV2ABGR420(3MLIB),
mlib_VideoColorYUV2ABGR422(3MLIB), mlib_VideoColorYUV2ARGB411(3MLIB),
mlib_VideoColorYUV2ARGB420(3MLIB), mlib_VideoColorYUV2ARGB422(3MLIB),
mlib_VideoColorYUV2ARGB444(3MLIB), mlib_VideoColorYUV2RGB411(3MLIB),
mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
mlib_VideoColorYUV2ARGB411 – YUV to RGB color conversion

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2ARGB411(mlib_u8 *argb, const mlib_u8 *y,
const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height,
mlib_s32 rgb_stride, mlib_s32 y_stride, mlib_s32 uv_stride);

Description

The mlib_VideoColorYUV2ARGB411() function performs YUV to RGB color conversion used in digital video compression in the 4:1:1 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 4-to-1 in only the horizontal direction in respect to the luminance component.

Parameters

The function takes the following arguments:

- argb Pointer to the destination packed ARGB image.
- y Pointer to the source Y component.
- u Pointer to the source U component.
- v Pointer to the source V component.
- width Width of the image.
- height Height of the image.
- rgb_stride Stride, in bytes, between adjacent rows in the destination image.
- y_stride Stride, in bytes, between adjacent rows in the Y component image.
- uv_stride Stride, in bytes, between adjacent rows in the U and V component images.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_VideoColorYUV2ABGR411(3MLIB), mlib_VideoColorYUV2ABGR420(3MLIB),
mlib_VideoColorYUV2ABGR422(3MLIB), mlib_VideoColorYUV2ABGR444(3MLIB),
mlib_VideoColorYUV2ARGB420(3MLIB), mlib_VideoColorYUV2ARGB444(3MLIB),
mlib_VideoColorYUV2ARGB422(3MLIB), mlib_VideoColorYUV2RGB444(3MLIB),
mlib_VideoColorYUV2RGB411(3MLIB),
mlib_VideoColorYUV2RGB411(3MLIB)

mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
# mlib_VideoColorYUV2ARGB420

## Name
mlib_VideoColorYUV2ARGB420 – YUV to RGB color conversion

## Synopsis
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2ARGB420(mlib_u8 *argb, const mlib_u8 *y,
                                       const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height,
                                       mlib_s32 rgb_stride, mlib_s32 y_stride, mlib_s32 uv_stride);
```

## Description
The `mlib_VideoColorYUV2ARGB420()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

## Parameters
The function takes the following arguments:
- `argb` Pointer to the destination packed ARGB image.
- `y` Pointer to the source Y component.
- `u` Pointer to the source U component.
- `v` Pointer to the source V component.
- `width` Width of the image.
- `height` Height of the image.
- `rgb_stride` Stride, in bytes, between adjacent rows in the destination image.
- `y_stride` Stride, in bytes, between adjacent rows in the Y component image.
- `uv_stride` Stride, in bytes, between adjacent rows in the U and V component images.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- `mlib_VideoColorYUV2ABGR411(3MLIB)`
- `mlib_VideoColorYUV2ABGR420(3MLIB)`
- `mlib_VideoColorYUV2ABGR422(3MLIB)`
- `mlib_VideoColorYUV2ABGR444(3MLIB)`
- `mlib_VideoColorYUV2ARGB411(3MLIB)`
- `mlib_VideoColorYUV2ARGB420(3MLIB)`
- `mlib_VideoColorYUV2ARGB422(3MLIB)`
- `mlib_VideoColorYUV2ARGB444(3MLIB)`
mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB), mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
The `mlib_VideoColorYUV2ARGB422` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:2 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in only the horizontal direction in respect to the luminance component.

The function takes the following arguments:
- `argb` Pointer to the destination packed ARGB image.
- `y` Pointer to the source Y component.
- `u` Pointer to the source U component.
- `v` Pointer to the source V component.
- `width` Width of the image.
- `height` Height of the image.
- `rgb_stride` Stride, in bytes, between adjacent rows in the destination image.
- `y_stride` Stride, in bytes, between adjacent rows in the Y component image.
- `uv_stride` Stride, in bytes, between adjacent rows in the U and V component images.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `mlib_VideoColorYUV2ABGR411(3MLIB)`, `mlib_VideoColorYUV2ABGR420(3MLIB)`, `mlib_VideoColorYUV2ABGR422(3MLIB)`, `mlib_VideoColorYUV2ABGR444(3MLIB)`, `mlib_VideoColorYUV2ARGB411(3MLIB)`, `mlib_VideoColorYUV2ARGB420(3MLIB)`, `mlib_VideoColorYUV2ARGB422(3MLIB)`, `mlib_VideoColorYUV2ARGB444(3MLIB)`.
mlib_VideoColorYUV2ARGB422(3MLIB)

mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
Name: mlib_VideoColorYUV2ARGB444 – YUV to RGB color conversion

Synopsis:
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2ARGB444(mlib_u8 *argb, const mlib_u8 *y,
                                       const mlib_u8 *u, const mlib_u8 *v,
                                       mlib_s32 width, mlib_s32 height,
                                       mlib_s32 rgb_stride, mlib_s32 yuv_stride);
```

Description:
The `mlib_VideoColorYUV2ARGB444()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:4:4 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components have the same resolution as the luminance component.

Parameters:
The function takes the following arguments:
- **argb**: Pointer to the destination packed ARGB image.
- **y**: Pointer to the source Y component.
- **u**: Pointer to the source U component.
- **v**: Pointer to the source V component.
- **width**: Width of the image.
- **height**: Height of the image.
- **rgb_stride**: Stride, in bytes, between adjacent rows in the destination image.
- **yuv_stride**: Stride, in bytes, between adjacent rows in the source image.

Return Values:
The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes:
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also:
- mlib_VideoColorYUV2ABGR411(3MLIB), mlib_VideoColorYUV2ABGR420(3MLIB),
- mlib_VideoColorYUV2ABGR422(3MLIB), mlib_VideoColorYUV2ABGR444(3MLIB),
- mlib_VideoColorYUV2ARGB411(3MLIB), mlib_VideoColorYUV2ARGB420(3MLIB),
- mlib_VideoColorYUV2ARGB422(3MLIB), mlib_VideoColorYUV2RGB411(3MLIB),
- mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
- mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
mlib_VideoColorYUV2RGB411 — YUV to RGB color conversion

**Synopsis**

```c
#include <mlib.h>

mlib_status mlib_VideoColorYUV2RGB411(mlib_u8 *
rgb, const mlib_u8 *
y, const mlib_u8 *
u, const mlib_u8 *
v, mlib_s32 width, mlib_s32 height,
mlib_s32 rgb_stride, mlib_s32 y_stride, mlib_s32 uv_stride);
```

**Description**

The `mlib_VideoColorYUV2RGB411()` function performs YUV to RGB color conversion used in digital video compression in the 4:1:1 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 4-to-1 in only the horizontal direction in respect to the luminance component.

**Parameters**

The function takes the following arguments:

- `rgb`  
  Pointer to the destination RGB image.
- `y`  
  Pointer to the source Y component.
- `u`  
  Pointer to the source U component.
- `v`  
  Pointer to the source V component.
- `width`  
  Width of the image.
- `height`  
  Height of the image.
- `rgb_stride`  
  Stride, in bytes, between adjacent rows in the destination image.
- `y_stride`  
  Stride, in bytes, between adjacent rows in the Y component image.
- `uv_stride`  
  Stride, in bytes, between adjacent rows in the U and V component images.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoColorYUV2ABGR411(3MLIB)`
- `mlib_VideoColorYUV2ABGR420(3MLIB)`
- `mlib_VideoColorYUV2ABGR422(3MLIB)`
- `mlib_VideoColorYUV2ABGR444(3MLIB)`
- `mlib_VideoColorYUV2ARGB411(3MLIB)`
- `mlib_VideoColorYUV2ARGB420(3MLIB)`
- `mlib_VideoColorYUV2ARGB422(3MLIB)`
- `mlib_VideoColorYUV2ARGB444(3MLIB)`
mlib_VideoColorYUV2RGB420(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
mlib_VideoColorYUV2RGB420 – YUV to RGB color conversion

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2RGB420(mlib_u8 *rgb, const mlib_u8 *y,
        const mlib_u8 *u, const mlib_u8 *v, mlib_s32 width, mlib_s32 height,
        mlib_s32 rgb_stride, mlib_s32 y_stride, mlib_s32 uv_stride);

Description

The mlib_VideoColorYUV2RGB420() function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:0 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in both the horizontal and vertical directions in respect to the luminance component.

Parameters

The function takes the following arguments:

rgb Pointer to the destination RGB image.
y Pointer to the source Y component.
u Pointer to the source U component.
v Pointer to the source V component.
width Width of the image.
height Height of the image.
rgb_stride Stride, in bytes, between adjacent rows in the destination image.
y_stride Stride, in bytes, between adjacent rows in the Y component image.
v_stride Stride, in bytes, between adjacent rows in the U and V component images.

Return Values

The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_VideoColorYUV2ABGR411(3MLIB), mlib_VideoColorYUV2ABGR420(3MLIB),
mlib_VideoColorYUV2ABGR422(3MLIB), mlib_VideoColorYUV2ABGR444(3MLIB),
mlib_VideoColorYUV2ARGB411(3MLIB), mlib_VideoColorYUV2ARGB420(3MLIB),
mlib_VideoColorYUV2ARGB422(3MLIB), mlib_VideoColorYUV2ARGB444(3MLIB),
mlib_VideoColorYUV2RGB411(3MLIB), mlib_VideoColorYUV2RGB422(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
The `mlib_VideoColorYUV2RGB422()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:2:2 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components are subsampled 2-to-1 in only the horizontal direction in respect to the luminance component.

### Parameters

- **rgb**
  - Pointer to the destination RGB image.
- **y**
  - Pointer to the source Y component.
- **u**
  - Pointer to the source U component.
- **v**
  - Pointer to the source V component.
- **width**
  - Width of the image.
- **height**
  - Height of the image.
- **rgb_stride**
  - Stride, in bytes, between adjacent rows in the destination image.
- **y_stride**
  - Stride, in bytes, between adjacent rows in the Y component image.
- **uv_stride**
  - Stride, in bytes, between adjacent rows in the U and V component images.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See section 5 for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_VideoColorYUV2ABGR411(3MLIB)`, `mlib_VideoColorYUV2ABGR420(3MLIB)`, `mlib_VideoColorYUV2ABGR422(3MLIB)`, `mlib_VideoColorYUV2ABGR444(3MLIB)`
- `mlib_VideoColorYUV2ARGB411(3MLIB)`, `mlib_VideoColorYUV2ARGB420(3MLIB)`, `mlib_VideoColorYUV2ARGB422(3MLIB)`, `mlib_VideoColorYUV2ARGB444(3MLIB)`
mlib_VideoColorYUV2RGB422(3MLIB)

mlib_VideoColorYUV2RGB411(3MLIB), mlib_VideoColorYUV2RGB420(3MLIB),
mlib_VideoColorYUV2RGB444(3MLIB), attributes(5)
# mlib_VideoColorYUV2RGB444

## Name
mlib_VideoColorYUV2RGB444 – YUV to RGB color conversion

## Synopsis
```
c C [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoColorYUV2RGB444(mlib_u8 *rgb, const mlib_u8 *y,
                                        const mlib_u8 *u, const mlib_u8 *v,
                                        mlib_s32 width, mlib_s32 height,
                                        mlib_s32 rgb_stride, mlib_s32 yuv_stride);
```

## Description
The `mlib_VideoColorYUV2RGB444()` function performs YUV to RGB color conversion used in MPEG1 and MPEG2 video compression in the 4:4:4 sequence.

The luminance component is stored in Y, the chrominance components are stored in U and V, respectively. The size of the chrominance image depends on the chroma format used by the sequence. In this sequence, the chrominance components have the same resolution as the luminance component.

## Parameters
The function takes the following arguments:
- `rgb` Pointer to the destination RGB image.
- `y` Pointer to the source Y component.
- `u` Pointer to the source U component.
- `v` Pointer to the source V component.
- `width` Width of the image.
- `height` Height of the image.
- `rgb_stride` Stride, in bytes, between adjacent rows in the destination image.
- `yuv_stride` Stride, in bytes, between adjacent rows in the source image.

## Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also
- mlib_VideoColorYUV2ABGR411(3MLIB), mlib_VideoColorYUV2ABGR420(3MLIB),
- mlib_VideoColorYUV2ABGR422(3MLIB), mlib_VideoColorYUV2ABGR444(3MLIB),
- mlib_VideoColorYUV2ARGB411(3MLIB), mlib_VideoColorYUV2ARGB420(3MLIB),
- mlib_VideoColorYUV2ARGB422(3MLIB), mlib_VideoColorYUV2ARGB444(3MLIB),
- mlib_VideoColorYUV2RGB411(3MLIB), mlib_VideoColorYUV2RGB420(3MLIB),
- mlib_VideoColorYUV2RGB422(3MLIB), attributes(5)
The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions $w$ and $h$. The U and V buffers have dimensions $w/4$ and $h$. Dimension $w$ is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The alpha values for this function work in the following fashion:

- If $A_{array}$ pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If $A_{array}$ pointer is NULL, the alpha values for every pixel are set to $A_{const}$.

The following equation is used:

$$
\begin{align*}
|R| &= \begin{bmatrix} 1.644 & 0.0000 & 1.9066 \end{bmatrix} \begin{bmatrix} Y \end{bmatrix} + 16.0000 \\
|G| &= \begin{bmatrix} 1.644 & -0.3920 & -0.8132 \end{bmatrix} \begin{bmatrix} U \end{bmatrix} - 128.0000 \\
|B| &= \begin{bmatrix} 1.644 & 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} V \end{bmatrix} + 128.0000
\end{align*}
$$

The function takes the following arguments:

- **ABGR**: Pointer to output buffer.
- **Y**: Pointer to Y input buffer.
- **U**: Pointer to U input buffer.
- **V**: Pointer to V input buffer.
- **A_{array}**: Array of alpha values.
- **A_{const}**: Constant alpha value.
- **w**: Image width in pixels.
- **h**: Image height in lines.
- **dlb**: Line bytes for output buffer.
- **yalb**: Line bytes for Y and alpha buffers.
- **uvlb**: Line bytes for U and V buffers.
## Return Values
None.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUYV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV444int_to_ABGRint(3MLIB),
mlib_VideoColorUYVY422int_to_ARGBint(3MLIB),
mlib_VideoColorUYVY422int_to_ABGRint(3MLIB),
mlib_VideoColorUYVY444int_to_ARGBint(3MLIB),
mlib_VideoColorUYVY444int_to_ABGRint(3MLIB),
mlib_VideoColorUYV444int_to_ARGBint(3MLIB),
mlib_VideoColorUYV444int_to_ABGRint(3MLIB),
attributes(5)
The function takes the following arguments:

- **ARGB** Pointer to output buffer.
- **Y** Pointer to Y input buffer.
- **U** Pointer to U input buffer.
- **V** Pointer to V input buffer.
- **A_array** Array of alpha values.
- **A_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **yalb** Linebytes for Y and alpha buffers.
- **uvlb** Linebytes for U and V buffers.

The function converts Y, U, V pixel streams into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have dimensions \( w/4 \) and \( h \). Dimension \( w \) is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The alpha values for this function work in the following fashion:

- If \( A\_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If \( A\_array \) pointer is NULL, the alpha values for every pixel are set to \( A\_const \).

The following equation is used:

\[
\begin{align*}
|R| &= \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \end{bmatrix} \begin{bmatrix} |Y| & 16.0000 \end{bmatrix} \\
|G| &= \begin{bmatrix} 1.1644 & -0.3920 & -0.8132 \end{bmatrix} \begin{bmatrix} |U| & 128.0000 \end{bmatrix} \\
|B| &= \begin{bmatrix} 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} |V| & 128.0000 \end{bmatrix}
\end{align*}
\]

**Parameters**

- **ARGB** Pointer to output buffer.
- **Y** Pointer to Y input buffer.
- **U** Pointer to U input buffer.
- **V** Pointer to V input buffer.
- **A_array** Array of alpha values.
- **A_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **yalb** Linebytes for Y and alpha buffers.
- **uvlb** Linebytes for U and V buffers.
null
Name  mlib_VideoColorYUV411seq_to_UYVY422int – convert YUV sequential to interleaved

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorYUV411seq_to_UYVY422int(mlib_u32 *UYVY,
const mlib_u8  *Y, const mlib_u8  *U,
const mlib_u8  *V, mlib_s32   w,
mlib_s32   h, mlib_s32  dlb,
mlib_s32  ylb, mlib_s32  uvlb);

Description  The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are
8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have
dimensions \( w/4 \) and \( h \). Dimension \( w \) is assumed to be a multiple of 4. In each row, every 4 Y
values use the same U and V values.

The following equation is used:

\[
UYVY[r][c/2] = (U[r][c/4] \ll 24) | (Y[r][c] \ll 16) | (V[r][c/4] \ll 8) | (Y[r][c+1])
\]

\[
UYVY[r][c/2+1] = (U[r][c/4] \ll 24) | (Y[r][c+2] \ll 16) | (V[r][c/4] \ll 8) | (Y[r][c+3])
\]

where \( r = 0, 2, 4, \ldots, h-2 \); and \( c = 0, 2, 4, \ldots, w-2 \).

Parameters  The function takes the following arguments:

- **UYVY**  Pointer to output buffer.
- **Y**  Pointer to Y input buffer.
- **U**  Pointer to U input buffer.
- **V**  Pointer to V input buffer.
- **w**  Image width in pixels.
- **h**  Image height in lines.
- **dlb**  Linebytes for UYVY buffer.
- **ylb**  Linebytes for Y buffer.
- **uvlb**  Linebytes for U and V buffers.
Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB),
mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB),
mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB),
mlib_VideoColorYUV420seq_to_UYVY422int(3MLIB),
mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB), attributes(5)
### Synopsis

```c
void mlib_VideoColorYUV411seq_to_YUYV422int(mlib_u32 *YUYV,
   const mlib_u8  *Y, const mlib_u8  *U, const mlib_u8  *V,
   mlib_s32  w, mlib_s32  h, mlib_s32 dlb,
   mlib_s32  ylb, mlib_s32 uvlb);
```

### Description

The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \(w\) and \(h\). The U and V buffers have dimensions \(w/4\) and \(h\). Dimension \(w\) is assumed to be a multiple of 4. In each row, every 4 Y values use the same U and V values.

The following equation is used:

\[
\begin{align*}
    Y_{UYV}[r][c/2] &= (Y[r][c] << 24) | \\
        & (U[r][c/4] << 16) | \\
        & (Y[r][c+1] << 8) | \\
        & (V[r][c/4])
    \\
    Y_{UYV}[r][c/2+1] &= (Y[r][c+2] << 24) | \\
        & (U[r][c/4] << 16) | \\
        & (Y[r][c+3] << 8) | \\
        & (V[r][c/4])
\end{align*}
\]

where \(r = 0, 2, 4, \ldots, h-2\); and \(c = 0, 2, 4, \ldots, w-2\).

### Parameters

The function takes the following arguments:

- **YUYV** Pointer to output buffer.
- **Y** Pointer to Y input buffer.
- **U** Pointer to U input buffer.
- **V** Pointer to V input buffer.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for YUYV buffer.
- **ylb** Linebytes for Y buffer.
- **uvlb** Linebytes for U and V buffers.

### Return Values

None.
Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VideoColorYUV420seq_to_YUVY422int(3MLIB),
            mlib_VideoColorYUV422seq_to_YUVY422int(3MLIB),
            mlib_VideoColorYUV420seq_to_UYVY422int(3MLIB),
            mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB),
            mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB), attributes(5)
The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have dimensions \( w/2 \) and \( h/2 \). Dimensions \( w \) and \( h \) are assumed to be even. Successive rows of the output buffer ABGR use successive rows of Y and the same rows of U and V.

The alpha values for this function work in the following fashion:

- If \( A_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If \( A_array \) pointer is NULL, the alpha values for every pixel are set to \( A_const \).

The following equation is used:

\[
\begin{bmatrix}
|R| = [1.1644 \\ 0.0000 \\ 1.5966] \cdot [|Y| - 16.0000] \\
|G| = [1.1644 \\ -0.3920 \\ -0.8132] \cdot [|U| - 128.0000] \\
|B| = [1.1644 \\ 2.0184 \\ 0.0000] \cdot [|V| - 128.0000]
\end{bmatrix}
\]

The function takes the following arguments:

- \( ABGR \) Pointer to output buffer.
- \( Y \) Pointer to Y input buffer.
- \( U \) Pointer to U input buffer.
- \( V \) Pointer to V input buffer.
- \( A_array \) Array of alpha values.
- \( A_const \) Constant alpha value.
- \( w \) Image width in pixels.
- \( h \) Image height in lines.
- \( dlb \) Linebytes for output buffer.
- \( yalb \) Linebytes for Y and alpha buffers.
- \( uvlb \) Linebytes for U and V buffers.
Return Values  None.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV411int_to_ARGBint(3MLIB),
mlib_VideoColorYUV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUV411int_to_ABGRint(3MLIB),
mlib_VideoColorYUV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUV444int_to_ABGRint(3MLIB),
mlib_VideoColorUYVY422int_to_ARGBint(3MLIB),
mlib_VideoColorUYVY422int_to_ABGRint(3MLIB),
mlib_VideoColorUYV444int_to_ARGBint(3MLIB),
mlib_VideoColorUYV444int_to_ABGRint(3MLIB),
mlib_VideoColorUYV444int_to_ABGRint(3MLIB), attributes(5)
mlib_VideoColorYUV420seq_to_ARGBint

Name
mlib_VideoColorYUV420seq_to_ARGBint – color convert YUV sequential to ARGB interleaved

Synopsis
#include <mlib.h>

void mlib_VideoColorYUV420seq_to_ARGBint(mlib_u32 *ARGB, 
   const mlib_u8 *Y, const mlib_u8 *U, const mlib_u8 *V, 
   const mlib_u8 *A_array, mlib_u8 A_const, 
   mlib_s32 w, mlib_s32 h, mlib_s32 dlb, 
   mlib_s32 yalb, mlib_s32 uvlb);

Description
The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \times h \). The U and V buffers have dimensions \( w/2 \times h/2 \). Dimensions \( w \) and \( h \) are assumed to be even. Successive rows of the output buffer ARGB use successive rows of Y and the same rows of U and V.

The alpha values for this function work in the following fashion:

- If \( A_{array} \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If \( A_{array} \) pointer is NULL, the alpha values for every pixel are set to \( A_{const} \).

The following equation is used:

\[
\begin{align*}
|R| &= [1.1644 \ 0.0000 \ 1.5966] \ [|Y| \ |16.0000|] \\
|G| &= [1.1644 -0.3920 -0.6132] \ [|U| - |128.0000|] \\
|B| &= [1.1644 \ 2.0184 \ 0.0000] \ [|V| - |128.0000|]
\end{align*}
\]

Parameters
The function takes the following arguments:

- \( ARGB \) Pointer to output buffer.
- \( Y \) Pointer to Y input buffer.
- \( U \) Pointer to U input buffer.
- \( V \) Pointer to V input buffer.
- \( A_{array} \) Array of alpha values.
- \( A_{const} \) Constant alpha value.
- \( w \) Image width in pixels.
- \( h \) Image height in lines.
- \( dlb \) Linebytes for output buffer.
- \( yalb \) Linebytes for Y and alpha buffers.
- \( uvlb \) Linebytes for U and V buffers.
Return Values
None.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
- mlib_VideoColorYUV411seq_to_ARGBint(3MLIB)
- mlib_VideoColorYUV422seq_to_ARGBint(3MLIB)
- mlib_VideoColorYUV444seq_to_ARGBint(3MLIB)
- mlib_VideoColorYUV420seq_to_ABGRint(3MLIB)
- mlib_VideoColorYUV411seq_to_ABGRint(3MLIB)
- mlib_VideoColorYUV422seq_to_ABGRint(3MLIB)
- mlib_VideoColorYUV444seq_to_ABGRint(3MLIB)
- mlib_VideoColorYUYV422int_to_ARGBint(3MLIB)
- mlib_VideoColorYUYV444int_to_ARGBint(3MLIB)
- mlib_VideoColorYUYV422int_to_ABGRint(3MLIB)
- mlib_VideoColorYUYV444int_to_ABGRint(3MLIB)
- attributes(5)
### Name
mlib_VideoColorYUV420seq_to_UYVY422int – convert YUV sequential to interleaved

### Synopsis
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorYUV420seq_to_UYVY422int(mlib_u32 *UYVY,
    const mlib_u8 *Y, const mlib_u8 *U, const mlib_u8 *V,
    mlib_s32 w, mlib_s32 h, mlib_s32 dlb,
    mlib_s32 ylb, mlib_s32 uvlb);
```

### Description
The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions $w$ and $h$. The U and V buffers have dimensions $w/2$ and $h/2$. Dimensions $w$ and $h$ are assumed to be even. Successive rows of the output buffer UYVY use successive rows of Y and the same rows of U and V.

The following equation is used:

\[
UYVY[r][c/2] = (U[r/2][c/2] \ll 24) | (Y[r][c] \ll 16) | (V[r/2][c/2] \ll 8) | (Y[r][c+1])
\]

\[
UYVY[r+1][c/2] = (U[r/2][c/2] \ll 24) | (Y[r+1][c] \ll 16) | (V[r/2][c/2] \ll 8) | (Y[r+1][c+1])
\]

where $r = 0, 2, 4, \ldots, h-2$; and $c = 0, 2, 4, \ldots, w-2$.

### Parameters
The function takes the following arguments:

- **UYVY**: Pointer to output buffer.
- **Y**: Pointer to Y input buffer.
- **U**: Pointer to U input buffer.
- **V**: Pointer to V input buffer.
- **w**: Image width in pixels.
- **h**: Image height in lines.
- **dlb**: Linebytes for UYVY buffer.
- **ylb**: Linebytes for Y buffer.
- **uvlb**: Linebytes for U and V buffers.

### Return Values
None.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoColorYUV420seq_to_UYVY422int(3MLIB),
          mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB),
          mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB),
          mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB),
          mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB),
          attributes(5)
The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have dimensions \( w/2 \) and \( h/2 \). Dimensions \( w \) and \( h \) are assumed to be even. Successive rows of the output buffer YUYV use successive rows of Y and the same rows of U and V.

The following equation is used:

\[
YUYV[r][c/2] = (Y[r][c] \ll 24) | (U[r/2][c/2] \ll 16) | (Y[r][c+1] \ll 8) | (V[r/2][c/2])
\]

\[
YUYV[r+1][c/2] = (Y[r+1][c] \ll 24) | (U[r/2][c/2] \ll 16) | (Y[r+1][c+1] \ll 8) | (V[r/2][c/2])
\]

where \( r = 0, 2, 4, \ldots, h-2 \); and \( c = 0, 2, 4, \ldots, w-2 \).

### Parameters

- **YUYV**  
  Pointer to output buffer.
- **Y**  
  Pointer to Y input buffer.
- **U**  
  Pointer to U input buffer.
- **V**  
  Pointer to V input buffer.
- **w**  
  Image width in pixels.
- **h**  
  Image height in lines.
- **dlb**  
  Linebytes for YUYV buffer.
- **ylb**  
  Linebytes for Y buffer.
- **uvlb**  
  Linebytes for U and V buffers.

### Return Values

None.
Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_VideoColorYUV411seq_to_YUYV422int(3MLIB), mlib_VideoColorYUV422seq_to_YUYV422int(3MLIB), mlib_VideoColorYUV420seq_to_UYVY422int(3MLIB), mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB), mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB), attributes(5)
mlib_VideoColorYUV422seq_to_ABGRint – color convert YUV sequential to ABGR interleaved

Synopsis

```c
#include <mlib.h>

void mlib_VideoColorYUV422seq_to_ABGRint(mlib_u32 *ABGR,
   const mlib_u8 *Y, const mlib_u8 *U, const mlib_u8 *V,
   const mlib_u8 *A_array, mlib_u8 A_const,
   mlib_s32 w, mlib_s32 h, mlib_s32 dlb, mlib_s32 yalb,
   mlib_s32 uvlb);
```

Description

The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have dimensions \( w/2 \) and \( h \). Dimensions \( w \) and \( h \) are assumed to be even. Similar to `mlib_VideoColorYUV420seq_to_ABGRint()` except U and V are not sampled in the h direction.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

\[
\begin{align*}
|R| & = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \end{bmatrix} \times \begin{bmatrix} Y \end{bmatrix} + 16.0000 \\
|G| & = \begin{bmatrix} 1.1644 & -0.3920 & -0.8132 \end{bmatrix} \times \begin{bmatrix} U \end{bmatrix} - 128.0000 \\
|B| & = \begin{bmatrix} 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \times \begin{bmatrix} V \end{bmatrix} - 128.0000
\end{align*}
\]

Parameters

The function takes the following arguments:

- **ABGR** Pointer to output buffer.
- **Y** Pointer to Y input buffer.
- **U** Pointer to U input buffer.
- **V** Pointer to V input buffer.
- **A_array** Array of alpha values.
- **A_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **yalb** Linebytes for Y and alpha buffers.
- **uvlb** Linebytes for U and V buffers.
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB)

**Return Values** None.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
- mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
- mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
- mlib_VideoColorYUYV444int_to_ARGBint(3MLIB),
- mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
- mlib_VideoColorYUYV444int_to_ABGRint(3MLIB),
- attributes(5)
The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \(w\) and \(h\). The U and V buffers have dimensions \(w/2\) and \(h\). Dimensions \(w\) and \(h\) are assumed to be even. Similar to \texttt{mlib_VideoColorYUV420seq_to_ARGBint()} except U and V are not sampled in the h direction.

The alpha values for this function work in the following fashion:

- If \texttt{A_array} pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If \texttt{A_array} pointer is NULL, the alpha values for every pixel are set to \texttt{A_const}.

The following equation is used:

\[
\begin{align*}
|R| &= \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \end{bmatrix} \cdot \begin{bmatrix} Y \end{bmatrix} + \begin{bmatrix} 16.0000 \end{bmatrix} \\
|G| &= \begin{bmatrix} 1.1644 & -0.3920 & -0.8132 \end{bmatrix} \cdot \begin{bmatrix} U \end{bmatrix} + \begin{bmatrix} 128.0000 \end{bmatrix} \\
|B| &= \begin{bmatrix} 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \cdot \begin{bmatrix} V \end{bmatrix} + \begin{bmatrix} 128.0000 \end{bmatrix}
\end{align*}
\]

Parameters

The function takes the following arguments:

- \texttt{ARGB} Pointer to output buffer.
- \texttt{Y} Pointer to Y input buffer.
- \texttt{U} Pointer to U input buffer.
- \texttt{V} Pointer to V input buffer.
- \texttt{A_array} Array of alpha values.
- \texttt{A_const} Constant alpha value.
- \texttt{w} Image width in pixels.
- \texttt{h} Image height in lines.
- \texttt{dlb} Linebytes for output buffer.
- \texttt{yalb} Linebytes for Y and alpha buffers.
uvlb Linebytes for U and V buffers.

Return Values None.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUV444int_to_ABGRint(3MLIB),
mlib_VideoColorUYVY422int_to_ARGBint(3MLIB),
mlib_VideoColorUYVY422int_to_ABGRint(3MLIB),
mlib_VideoColorUYV444int_to_ARGBint(3MLIB),
mlib_VideoColorUYV444int_to_ABGRint(3MLIB),
mlib_VideoColorUYV444int_to_ABGRint(3MLIB),
attributes(5)
**Name**
mlib_VideoColorYUV422seq_to_UYVY422int – convert YUV sequential to interleaved

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
void mlib_VideoColorYUV422seq_to_UYVY422int(mlib_u32 *UYVY,
    const mlib_u8  *Y, const mlib_u8  *U, const mlib_u8  *V,
    mlib_s32 w, mlib_s32 h, mlib_s32 dlb,
    mlib_s32 ylb, mlib_s32 uvlb);
```

**Description**
The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have dimensions \( w/2 \) and \( h \). Dimensions \( w \) and \( h \) are assumed to be even. Similar to `mlib_VideoColorYUV420seq_to_UYVY422int()` except U and V are not sampled in the h direction.

The following equation is used:

\[
UYVY[r][c/2] = (U[r][c/2] << 24) | (Y[r][c] << 16) | (V[r][c/2] << 8) | (Y[r][c+1])
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

**Parameters**
The function takes the following arguments:

- \( UYVY \) Pointer to output buffer.
- \( Y \) Pointer to Y input buffer.
- \( U \) Pointer to U input buffer.
- \( V \) Pointer to V input buffer.
- \( w \) Image width in pixels.
- \( h \) Image height in lines.
- \( dlb \) Linebytes for UYVY buffer.
- \( ylb \) Linebytes for Y buffer.
- \( uvlb \) Linebytes for U and V buffers.

**Return Values**
None.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>

Multimedia Library Functions - Part 7 1373
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoColorYUV420seq_to_UYVY422int(3MLIB)`
- `mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB)`
- `mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB)`
- `mlib_VideoColorYUV420seq_to_UYYV422int(3MLIB)`
- `mlib_VideoColorYUV411seq_to_UYYV422int(3MLIB)`
- `attributes(5)`
mlib_VideoColorYUV422seq_to_YUYV422int – convert YUV sequential to interleaved

Synopsis

```c
void mlib_VideoColorYUV422seq_to_YUYV422int(mlib_u32 *YUYV, 
   const mlib_u8 *Y, const mlib_u8 *U, const mlib_u8 *V, 
   mlib_s32 w, mlib_s32 h, mlib_s32 dlb, 
   mlib_s32 ylb, mlib_s32 uvlb);
```

Description

The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y buffer has dimensions \( w \) and \( h \). The U and V buffers have dimensions \( w/2 \) and \( h \). Dimensions \( w \) and \( h \) are assumed to be even. Similar to `mlib_VideoColorYUV420seq_to_YUYV422int()` except U and V are not sampled in the \( h \) direction.

The following equation is used:

\[
YUYV[r][c/2] = (Y[r][c] << 24) | \\
(U[r][c/2] << 16) | \\
(Y[r][c+1] << 8) | \\
(V[r][c/2])
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

Parameters

The function takes the following arguments:

- **YUYV** Pointer to output buffer.
- **Y** Pointer to Y input buffer.
- **U** Pointer to U input buffer.
- **V** Pointer to V input buffer.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for YUYV buffer.
- **ylb** Linebytes for Y buffer.
- **uvlb** Linebytes for U and V buffers.

Return Values

None.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>Attribute Type</td>
<td>Attribute Value</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoColorYUV420seq_to_YUYV422int(3MLIB)`,
- `mlib_VideoColorYUV411seq_to_YUYV422int(3MLIB)`,
- `mlib_VideoColorYUV420seq_to_UYVY422int(3MLIB)`,
- `mlib_VideoColorYUV411seq_to_UYVY422int(3MLIB)`,
- `mlib_VideoColorYUV422seq_to_UYVY422int(3MLIB)`,
- `attributes(5)`
**Name**  
mlib_VideoColorYUV444int_to_ABGRint – color convert YUV interleaved to ABGR interleaved

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  

```c
void mlib_VideoColorYUV444int_to_ABGRint(mlib_u32 *ABGR,
    const mlib_u8 *YUV, const mlib_u8 *A_array, mlib_u8 A_const,
    mlib_s32 w, mlib_s32 h, mlib_s32 dlb,
    mlib_s32 slb, mlib_s32 alb);
```

**Description**  
The YUV pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions `w` and `h`.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not `NULL`, the values are taken from there. It has to have the same dimensions as the `Y` buffer.
- If `A_array` pointer is `NULL`, the alpha values for every pixel are set to `A_const`.

The following equation is used:

\[
\begin{bmatrix}
|R| & |G| & |B|
\end{bmatrix} =
\begin{bmatrix}
1.644 & 0.392 & 2.0184 \\
1.644 & -0.392 & 0.0000 \\
1.644 & 0.000 & 0.5872
\end{bmatrix}
\begin{bmatrix}
16.0 & 128.0 & 0.0
\end{bmatrix}
\]

**Parameters**  
The function takes the following arguments:

- **ABGR**  
    Pointer to output buffer.
- **YUV**  
    Pointer to Y input buffer.
- **A_array**  
    Array of alpha values.
- **A_const**  
    Constant alpha value.
- **w**  
    Image width in pixels.
- **h**  
    Image height in lines.
- **dlb**  
    Linebytes for output buffer.
- **slb**  
    Linebytes for input buffer.
- **alb**  
    Linebytes for alpha buffer.

**Return Values**  
None.

**Attributes**  
See `attributes(5)` for descriptions of the following attributes:
**ATTRIBUTE TYPE** | **ATTRIBUTE VALUE**
--- | ---
Interface Stability | Committed
MT-Level | MT-Safe

See Also
- `mlib_VideoColorYUV444int_to_ABGRint(3MLIB)`
- `mlib_VideoColorYUV444int_to_ARGBint(3MLIB)`
- `mlib_VideoColorYUV422seq_to_ARGBint(3MLIB)`
- `mlib_VideoColorYUV420seq_to_ARGBint(3MLIB)`
- `mlib_VideoColorYUV420seq_to_ABGRint(3MLIB)`
- `mlib_VideoColorYUV422seq_to_ABGRint(3MLIB)`
- `mlib_VideoColorYUV411seq_to_ARGBint(3MLIB)`
- `mlib_VideoColorYUV411seq_to_ABGRint(3MLIB)`
- `mlib_VideoColorYUYV422int_to_ARGBint(3MLIB)`
- `mlib_VideoColorYUYV422int_to_ABGRint(3MLIB)`
- `mlib_VideoColorUYVY422int_to_ARGBint(3MLIB)`
- `mlib_VideoColorUYVY422int_to_ABGRint(3MLIB)`
- `mlib_VideoColorUYV444int_to_ARGBint(3MLIB)`
- `mlib_VideoColorUYV444int_to_ABGRint(3MLIB)`
- `attributes(5)`
The YUV pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions \(w\) and \(h\).

The alpha values for this function work in the following fashion:

- If \(A_{\text{array}}\) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \(Y\) buffer.
- If \(A_{\text{array}}\) pointer is NULL, the alpha values for every pixel are set to \(A_{\const}\).

The following equation is used:

\[
\begin{bmatrix}
R \\
G \\
B \\
\end{bmatrix} =
\begin{bmatrix}
1.1644 & 0.0000 & 1.5966 \\
1.1644 & -0.3920 & -0.8132 \\
1.1644 & 2.0184 & 0.0000 \\
\end{bmatrix} \times
\begin{bmatrix}
Y \\
U \\
V \\
\end{bmatrix} - \begin{bmatrix}
16.0000 \\
128.0000 \\
128.0000 \\
\end{bmatrix}
\]

The function takes the following arguments:

- \(ARGB\) Pointer to output buffer.
- \(YUV\) Pointer to \(Y\) input buffer.
- \(A_{\text{array}}\) Array of alpha values.
- \(A_{\const}\) Constant alpha value.
- \(w\) Image width in pixels.
- \(h\) Image height in lines.
- \(dlb\) Linebytes for output buffer.
- \(slb\) Linebytes for input buffer.
- \(alb\) Linebytes for alpha buffer.

None.

See attributes(5) for descriptions of the following attributes:
mlib_VideoColorYUV444int_to_ARGBint(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV444seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
mlib_VideoColorYUYV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUV444int_to_ABGRint(3MLIB),
mlib_VideoColorUYVY422int_to_ARGBint(3MLIB),
mlib_VideoColorUYVY422int_to_ABGRint(3MLIB),
mlib_VideoColorUYV444int_to_ARGBint(3MLIB),
mlib_VideoColorUYV444int_to_ABGRint(3MLIB),
attributes(5)
The YUV pixel stream is broken apart and recombined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The YUV buffer has dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values. The sequence of values in the input stream is \( Y[r][c], U[r][c], V[r][c], Y[r][c+1], U[r][c+1], V[r][c+1], \ldots \)

The following equation is used:

\[
UYVY[r][c/2] = (((U[r][c] + U[r][c+1]) / 2) \ll 24) | \\
(Y[r][c] \ll 16) | \\
(((V[r][c] + V[r][c+1]) / 2) \ll 8) | \\
(Y[r][c+1])
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

**Parameters**

The function takes the following arguments:

- **UYVY** Pointer to output buffer.
- **YUV** Pointer to input buffer.
- **\( w \)** Image width in pixels.
- **\( h \)** Image height in lines.
- **\( dlb \)** Linebytes for output buffer.
- **\( slb \)** Linebytes for input buffer.

**Return Values** None.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- mlib_VideoColorYUV444seq_to_YUYV422int(3MLIB)
- mlib_VideoColorYUV444int_to_YUYV422int(3MLIB)
- mlib_VideoColorYUV444seq_to_UYVY422int(3MLIB)
mlib_VideoColorYUYV444int_to_UYVY422int(3MLIB)
mlib_VideoColorYUV444int_to_UYVY422int(3MLIB),
mlib_VideoColorUYV444int_to_UYVY422int(3MLIB), attributes(5)
#include <mlib.h>

void mlib_VideoColorYUV444int_to_YUYV422int(mlib_u32 *YUYV, const mlib_u8 *YUV, mlib_s32 w, mlib_s32 h, mlib_s32 dl, mlib_s32 sl);

The YUV pixel stream is broken apart and recombined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The YUV buffer has dimensions $w$ and $h$. Dimension $w$ is assumed to be a multiple of 2. Adjacent $U$ and $V$ values are averaged to get the output $U$ and $V$ values. The sequence of values in the input stream is $Y[r][c], U[r][c], V[r][c], Y[r][c+1], U[r][c+1], V[r][c+1], ...$

The following equation is used:

$$YUYV[r][c/2] = (Y[r][c] \ll 24) | (((U[r][c] + U[r][c+1]) / 2) \ll 16) | (Y[r][c+1] \ll 8) | (((V[r][c] + V[r][c+1]) / 2))$$

where $r = 0, 1, 2, ..., h-1$; and $c = 0, 2, 4, ..., w-2$.

The function takes the following arguments:

- **YUYV**: Pointer to output buffer.
- **YUV**: Pointer to input buffer.
- **$w$**: Image width in pixels.
- **$h$**: Image height in lines.
- **dl**: Linebytes for output buffer.
- **sl**: Linebytes for input buffers.

None.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- mlib_VideoColorYUV444seq_to_YUYV422int(3MLIB)
- mlib_VideoColorYUV444seq_to_UYVY422int(3MLIB)
- mlib_VideoColorYUV444int_to_UYVY422int(3MLIB)
mlib_VideoColorYUV444int_to_YUYV422int(3MLIB),
mlib_VideoColorYUV444int_to_UYVY422int(3MLIB), attributes(5)
mlib_VideoColorYUV444seq_to_ABGRint – color convert YUV sequential to ABGR interleaved

**Synopsis**
```c
#include <mlib.h>

void mlib_VideoColorYUV444seq_to_ABGRint(mlib_u32 *ABGR, const mlib_u8 *Y,  
const mlib_u8 *U, const mlib_u8 *V, const mlib_u8 *A_array,  
mlib_u8 *A_const, mlib_s32 w, mlib_s32 h, mlib_s32 dlb, mlib_s32 slb);
```

**Description**
The Y, U, V pixel streams are converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions \(w\) and \(h\).

The alpha values for this function work in the following fashion:
- If \(A\_array\) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If \(A\_array\) pointer is NULL, the alpha values for every pixel are set to \(A\_const\).

The following equation is used:

\[
\begin{align*}
[R] & = \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \end{bmatrix} \begin{bmatrix} Y \end{bmatrix} - 16.0000 \\
[G] & = \begin{bmatrix} 1.1644 & -0.3920 & -0.8132 \end{bmatrix} \begin{bmatrix} U \end{bmatrix} - 128.0000 \\
[B] & = \begin{bmatrix} 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} V \end{bmatrix} - 128.0000
\end{align*}
\]

**Parameters**
The function takes the following arguments:
- \(ABGR\) Pointer to output buffer.
- \(Y\) Pointer to Y input buffer.
- \(U\) Pointer to U input buffer.
- \(V\) Pointer to V input buffer.
- \(A\_array\) Array of alpha values.
- \(A\_const\) Constant alpha value.
- \(w\) Image width in pixels.
- \(h\) Image height in lines.
- \(dlb\) Linebytes for output buffer.
- \(slb\) Linebytes for input buffers.

**Return Values**
None.

**Attributes**
See attributes(5) for descriptions of the following attributes:
## mlib_VideoColorYUV444seq_to_ABGRint

### Attributes

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoColorYUV420seq_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV411seq_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV422seq_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV444seq_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV420seq_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV411seq_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV422seq_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV444seq_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV420int_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV411int_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV422int_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV444int_to_ARGBint(3MLIB)`,
- `mlib_VideoColorYUV420int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV411int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV422int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorYUV444int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorUYVY422int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorUYVY422int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorUYVY444int_to_ABGRint(3MLIB)`,
- `mlib_VideoColorUYVY444int_to_ABGRint(3MLIB)`,
# mlib_VideoColorYUV444seq_to_ARGBint

## Synopsis
```c
#include <mlib.h>

void mlib_VideoColorYUV444seq_to_ARGBint(  
    mlib_u32 *ARGB,  
    const mlib_u8 *Y,  
    const mlib_u8 *U,  
    const mlib_u8 *V,  
    const mlib_u8 *A_array,  
    mlib_u8 A_const,  
    mlib_s32 w,  
    mlib_s32 h,  
    mlib_s32 dl,  
    mlib_s32 sl);
```

## Description
The Y, U, V pixel streams are converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. All buffers have dimensions `w` and `h`.

The alpha values for this function work in the following fashion:

- If `A_array` pointer is not NULL, the values are taken from there. It has to have the same dimensions as the Y buffer.
- If `A_array` pointer is NULL, the alpha values for every pixel are set to `A_const`.

The following equation is used:

\[
\begin{align*}
|R| &= \begin{bmatrix} 1.1644 & 0.0000 & 1.5966 \end{bmatrix} \begin{bmatrix} |Y| & 16.0000 \end{bmatrix} \\
|G| &= \begin{bmatrix} 1.1644 & -0.3920 & -0.8132 \end{bmatrix} \begin{bmatrix} |U| - 128.0000 \end{bmatrix} \\
|B| &= \begin{bmatrix} 1.1644 & 2.0184 & 0.0000 \end{bmatrix} \begin{bmatrix} |V| - 128.0000 \end{bmatrix}
\end{align*}
\]

## Parameters
The function takes the following arguments:

- `ARGB` Pointer to output buffer.
- `Y` Pointer to Y input buffer.
- `U` Pointer to U input buffer.
- `V` Pointer to V input buffer.
- `A_array` Array of alpha values.
- `A_const` Constant alpha value.
- `w` Image width in pixels.
- `h` Image height in lines.
- `dl` Linebytes for output buffer.
- `sl` Linebytes for input buffers.

## Return Values
None.

## Attributes
See `attributes(5)` for descriptions of the following attributes:
**mlib_VideoColorYUV444seq_to_ARGBint(3MLIB)**

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

mlib_VideoColorYUV420seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV411seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV422seq_to_ARGBint(3MLIB),
mlib_VideoColorYUV420seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV411seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV444seq_to_ABGRint(3MLIB),
mlib_VideoColorYUV422int_to_ARGBint(3MLIB),
mlib_VideoColorYUV444int_to_ARGBint(3MLIB),
mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUV444int_to_ABGRint(3MLIB),
mlib_VideoColorYUYV422int_to_ABGRint(3MLIB),
mlib_VideoColorYUYV444int_to_ABGRint(3MLIB),
mlib_VideoColorYUYV444int_to_ABGRint(3MLIB),attributes(5)
mlib_VideoColorYUV444seq_to_UYVY422int – convert YUV sequential to interleaved with subsampling

Synopsis

```c
#include <mlib.h>

void mlib_VideoColorYUV444seq_to_UYVY422int(mlib_u32 *UYVY,
    const mlib_u8 *Y, const mlib_u8 *U, const mlib_u8 *V, mlib_s32 w,
    mlib_s32 h, mlib_s32 dlbl, mlib_s32 slbl);
```

Description

The Y, U, V pixel streams are combined into a UYVY pixel stream. All pixel components are 8-bit unsigned integers. The Y, U, and V buffers have dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values.

The following equation is used:

\[
UYVY[r][c/2] = (((U[r][c] + U[r][c+1]) / 2) << 24) \mid
(Y[r][c] << 16) \mid
(((V[r][c] + V[r][c+1]) / 2) << 8) \mid
(Y[r][c+1])
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

Parameters

- **UYVY**: Pointer to output buffer.
- **Y**: Pointer to Y input buffer.
- **U**: Pointer to U input buffer.
- **V**: Pointer to V input buffer.
- **w**: Image width in pixels.
- **h**: Image height in lines.
- **dlbl**: Linebytes for output buffer.
- **slbl**: Linebytes for input buffers.

Return Values

None.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VideoColorYUV444seq_to_UYVY422int(3MLIB),
         mlib_VideoColorYUV444int_to_UYVY422int(3MLIB),
         mlib_VideoColorYUV444int_to_UYVY422int(3MLIB), attributes(5)
mlib_VideoColorYUV444seq_to_YUYV422int – convert YUV sequential to interleaved with subsampling

```
cc [ flag... ] file... -Imlib [ library... ]
#include <mlib.h>

void mlib_VideoColorYUV444seq_to_YUYV422int(mlib_u32 *YUYV, const mlib_u8 *Y, const mlib_u8 *U, const mlib_u8 *V, mlib_s32 w, mlib_s32 h, mlib_s32 dl, mlib_s32 sl);
```

**Description**
The Y, U, V pixel streams are combined into a YUYV pixel stream. All pixel components are 8-bit unsigned integers. The Y, U, and V buffers have dimensions \( w \) and \( h \). Dimension \( w \) is assumed to be a multiple of 2. Adjacent U and V values are averaged to get the output U and V values.

The following equation is used:

\[
YUYV[r][c/2] = (Y[r][c] \ll 24) | \\
((U[r][c] + U[r][c+1]) / 2) \ll 16 | \\
(Y[r][c+1] \ll 8) | \\
((V[r][c] + V[r][c+1]) / 2))
\]

where \( r = 0, 1, 2, \ldots, h-1 \); and \( c = 0, 2, 4, \ldots, w-2 \).

**Parameters**
The function takes the following arguments:

- \( YUYV \)  Pointer to output buffer.
- \( Y \)  Pointer to Y input buffer.
- \( U \)  Pointer to U input buffer.
- \( V \)  Pointer to V input buffer.
- \( w \)  Image width in pixels.
- \( h \)  Image height in lines.
- \( dl \)  Linebytes for output buffer.
- \( sl \)  Linebytes for input buffers.

**Return Values**
None.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  

  - mlib_VideoColorYUV444seq_to_UYVY422int(3MLIB),
  - mlib_VideoColorYUV444int_to_YUYV422int(3MLIB),
  - mlib_VideoColorYUV444int_to_UYVY422int(3MLIB),
  - attributes(5)
Name: mlib_VideoColorYUYV422int_to_ABGRint – color convert YUYV interleaved to ABGR interleaved

Synopsis: cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

void mlib_VideoColorYUYV422int_to_ABGRint(mlib_u32 *ABGR,
const mlib_u32 *YUYV, const mlib_u8 *A_array, mlib_u8 A_const,
mlib_s32 w, mlib_s32 h, mlib_s32 dlb, mlib_s32 slb, mlib_s32 alb);

Description: The YUYV pixel stream is converted into an ABGR pixel stream. All pixel components are 8-bit unsigned integers. The YUYV buffer has dimensions \( \frac{w}{2} \) and \( h \). The ABGR buffer has dimensions \( w \) and \( h \). Dimensions \( w \) is assumed to be even.

The alpha values for this function work in the following fashion:

- If \( A\_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \( Y \) buffer.
- If \( A\_array \) pointer is NULL, the alpha values for every pixel are set to \( A\_const \).

The following equation is used:

\[
\begin{align*}
|R| &= |1.1644 - 0.0000 - 1.5966| \times [\begin{bmatrix} Y \end{bmatrix} - 16.0000] \\
|G| &= |1.1644 - 0.3920 - 0.8132| \times [\begin{bmatrix} U \end{bmatrix} - 128.0000] \\
|B| &= |1.1644 2.0184 0.0000| \times [\begin{bmatrix} V \end{bmatrix} - 128.0000]
\end{align*}
\]

Parameters: The function takes the following arguments:

- \( ABGR \): Pointer to output buffer.
- \( YUYV \): Pointer to Y input buffer.
- \( A\_array \): Array of alpha values.
- \( A\_const \): Constant alpha value.
- \( w \): Image width in pixels.
- \( h \): Image height in lines.
- \( dlb \): Linebytes for output buffer.
- \( slb \): Linebytes for input buffer.
- \( alb \): Linebytes for alpha buffer.

Return Values: None.

Attributes: See attributes(5) for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VideoColorYUYV422int_to_ABGRint(3MLIB), mlib_VideoColorYUV420seq_to_ARGBint(3MLIB), mlib_VideoColorYUV411seq_to_ARGBint(3MLIB), mlib_VideoColorYUV422seq_to_ARGBint(3MLIB), mlib_VideoColorYUV444seq_to_ARGBint(3MLIB), mlib_VideoColorYUYV422int_to_ABGRint(3MLIB), mlib_VideoColorYUV444int_to_ARGBint(3MLIB), mlib_VideoColorYUYV422int_to_ABGRint(3MLIB), mlib_VideoColorYUV444int_to_ARGBint(3MLIB), mlib_VideoColorYUYV422int_to_ABGRint(3MLIB), mlib_VideoColorYUYV422int_to_ABGRint(3MLIB), mlib_VideoColorYUV444int_to_ABGRint(3MLIB), attributes(5)
mlib_VideoColorYUYV422int_to_ARGBint – color convert YUYV interleaved to ARGB interleaved

Synopsis

```c
#include <mlib.h>

void mlib_VideoColorYUYV422int_to_ARGBint(mlib_u32 *ARGB,
       const mlib_u32 *YUYV, const mlib_u8 *A_array, mlib_u8 A_const,
       mlib_s32 w, mlib_s32 h, mlib_s32 dlb, mlib_s32 slb, mlib_s32 alb);
```

Description

The YUYV pixel stream is converted into an ARGB pixel stream. All pixel components are 8-bit unsigned integers. The YUYV buffer has dimensions \( w/2 \) and \( h \). The ABGR buffer has dimensions \( w \) and \( h \). Dimensions \( w \) is assumed to be even.

The alpha values for this function work in the following fashion:

- If \( A\_array \) pointer is not NULL, the values are taken from there. It has to have the same dimensions as the \( Y \) buffer.
- If \( A\_array \) pointer is NULL, the alpha values for every pixel are set to \( A\_const \).

The following equation is used:

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} = \begin{bmatrix}
1.1644 & 0.0000 & 1.5966 \\
1.1644 & -0.3920 & -0.8132 \\
1.1644 & 2.0184 & 0.0000
\end{bmatrix} \begin{bmatrix}
Y \\
U \\
V
\end{bmatrix}
\]

Parameters

- **ARGB** Pointer to output buffer.
- **YUYV** Pointer to Y input buffer.
- **A\_array** Array of alpha values.
- **A\_const** Constant alpha value.
- **w** Image width in pixels.
- **h** Image height in lines.
- **dlb** Linebytes for output buffer.
- **slb** Linebytes for input buffer.
- **alb** Linebytes for alpha buffer.

Return Values

None.

Attributes

See `attributes(5)` for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoColorYUYV422int_to_ARGBInt(3MLIB)`
- `mlib_VideoColorYUYV444int_to_ARGBInt(3MLIB)`
- `mlib_VideoColorYUYV422int_to_ABGRInt(3MLIB)`
- `mlib_VideoColorYUYV444int_to_ABGRInt(3MLIB)`
- `mlib_VideoColorYUYV444int_to_ARGBInt(3MLIB)`, attributes(5)
Name  mlib_VideoCopyRefAve_U8_U8_16x16, mlib_VideoCopyRefAve_U8_U8_16x8, 
      mlib_VideoCopyRefAve_U8_U8_8x16, mlib_VideoCopyRefAve_U8_U8_8x8, 
      mlib_VideoCopyRefAve_U8_U8_8x4 – copies and averages a block from the reference block 
      to the current block

Synopsis  cc [ flag... ] file... -lmlib [ library... ] 
          #include <mlib.h>

          mlib_status mlib_VideoCopyRefAve_U8_U8_16x16(mlib_u8 *curr_block, 
          const mlib_u8 *ref_block, mlib_s32 stride);
          mlib_status mlib_VideoCopyRefAve_U8_U8_16x8(mlib_u8 *curr_block, 
          const mlib_u8 *ref_block, mlib_s32 stride);
          mlib_status mlib_VideoCopyRefAve_U8_U8_8x16(mlib_u8 *curr_block, 
          const mlib_u8 *ref_block, mlib_s32 stride);
          mlib_status mlib_VideoCopyRefAve_U8_U8_8x8(mlib_u8 *curr_block, 
          const mlib_u8 *ref_block, mlib_s32 stride);
          mlib_status mlib_VideoCopyRefAve_U8_U8_8x4(mlib_u8 *curr_block, 
          const mlib_u8 *ref_block, mlib_s32 stride);

Description  Each of these functions copies and averages a block from the reference block to the current 
block. The stride applies to both the input reference block and the current block.

Parameters  Each of the functions takes the following arguments:

          curr_block          Pointer to the current block. curr_block must be 8-byte aligned.
          ref_block          Pointer to the reference block.
          stride            Stride, in bytes, between adjacent rows in both the current block and the 
                             reference block. stride must be a multiple of eight.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB), 
          mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB), 
          mlib_VideoCopyRef_U8_U8_8x16(3MLIB), mlib_VideoCopyRef_U8_U8_8x8(3MLIB), 
          mlib_VideoCopyRef_U8_U8_8x4(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB), 
          mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB), 
          mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_8x8(3MLIB), 
          mlib_VideoInterpAveXY_U8_U8_8x4(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)

mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
mlib_VideoCopyRefAve_U8_U8 – copies and averages a block from the reference block to the current block.

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoCopyRefAve_U8_U8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
mlib_s32 stride);

Description
The mlib_VideoCopyRefAve_U8_U8() function copies and averages a block from the reference block to the current block. The stride applies to both the input reference block and the current block.

Parameters
The function takes the following arguments:

curr_block Pointer to the current block. curr_block must be 8-byte aligned.
ref_block Pointer to the reference block.
width Width of the blocks
height Height of the blocks.
stride Stride, in bytes, between adjacent rows in both the current block and the reference block. stride must be a multiple of eight.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveX_U8_U8(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveY_U8_U8(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Name
mlib_VideoCopyRef_S16_U8_16x16, mlib_VideoCopyRef_S16_U8_16x8,
mlib_VideoCopyRef_S16_U8_8x16, mlib_VideoCopyRef_S16_U8_8x8,
mlib_VideoCopyRef_S16_U8_8x4 – copies a block from the reference block to the current block

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoCopyRef_S16_U8_16x16(mlib_s16 *mc_block,
const mlib_u8 *ref_block, mlib_s32 stride);
mlib_status mlib_VideoCopyRef_S16_U8_16x8(mlib_s16 *mc_block,
const mlib_u8 *ref_block, mlib_s32 stride);
mlib_status mlib_VideoCopyRef_S16_U8_8x16(mlib_s16 *mc_block,
const mlib_u8 *ref_block, mlib_s32 stride);
mlib_status mlib_VideoCopyRef_S16_U8_8x8(mlib_s16 *mc_block,
const mlib_u8 *ref_block, mlib_s32 stride);
mlib_status mlib_VideoCopyRef_S16_U8_8x4(mlib_s16 *mc_block,
const mlib_u8 *ref_block, mlib_s32 stride);

Description
Each of these functions copies a block from the reference block to the motion-compensated reference block. The stride applies to only the input reference block.

Parameters
Each of the functions takes the following arguments:

- **mc_block** Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.
- **ref_block** Pointer to the reference block.
- **stride** Stride, in bytes, between adjacent rows in the reference block. stride must be a multiple of eight.

Return Values
Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_U8_U8(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8(3MLIB), mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB)

mlib_VideoInterpAveX_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveXU8_U8_16x16(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB),
attributes(5)
**Name**
mlib_VideoCopyRef_S16_U8 – copies a block from the reference block to the current block

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoCopyRef_S16_U8(mlib_s16 *mc_block,
    const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
    mlib_s32 stride);

**Description**
The mlib_VideoCopyRef_S16_U8() function copies a block from the reference block to the motion-compensated reference block. The stride applies to only the input reference block.

**Parameters**
The function takes the following arguments:
- **mc_block** Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.
- **ref_block** Pointer to the reference block.
- **width** Width of the blocks.
- **height** Height of the blocks.
- **stride** Stride, in bytes, between adjacent rows in the reference block. stride must be a multiple of eight.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_UC8_U8(3MLIB), mlib_VideoCopyRef_UC8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_UC8_U8(3MLIB), mlib_VideoCopyRefAve_UC8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB), mlib_VideoH263OverlappedMC_UC8_U8(3MLIB), mlib_VideoInterpAveX_UC8_U8(3MLIB), mlib_VideoInterpAveXY_UC8_U8(3MLIB), mlib_VideoInterpAveY_UC8_U8(3MLIB), mlib_VideoInterpAveY_UC8_U8_16x16(3MLIB), mlib_VideoInterpXY_UC8_U8(3MLIB), mlib_VideoInterpXY_UC8_U8_16x16(3MLIB), mlib_VideoInterp_UC8_S16(3MLIB), mlib_VideoInterp_UC8_S16_16x16(3MLIB), mlib_VideoInterp_UC8_UC8(3MLIB), mlib_VideoInterp_UC8_UC8_16x16(3MLIB), mlib_VideoInterpY_UC8_UC8(3MLIB), mlib_VideoInterpY_UC8_UC8_16x16(3MLIB), mlib_VideoInterpY_UC8_U8(3MLIB), mlib_VideoInterpY_UC8_U8_16x16(3MLIB),
mlib_VideoCopyRef_S16_U8(3MLIB)

mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Name  mlib_VideoCopyRef_U8_U8_16x16, mlib_VideoCopyRef_U8_U8_16x8,  
mlib_VideoCopyRef_U8_U8_8x16, mlib_VideoCopyRef_U8_U8_8x8,  
mlib_VideoCopyRef_U8_U8_8x4 – copies an 8x8 block from the reference block to the  
current block

Synopsis  cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_VideoCopyRef_U8_U8_16x16(mlib_u8 *curr_block,  
const mlib_u8 *ref_block, mlib_s32 stride);

mlib_status mlib_VideoCopyRef_U8_U8_16x8(mlib_u8 *curr_block,  
const mlib_u8 *ref_block, mlib_s32 stride);

mlib_status mlib_VideoCopyRef_U8_U8_8x16(mlib_u8 *curr_block,  
const mlib_u8 *ref_block, mlib_s32 stride);

mlib_status mlib_VideoCopyRef_U8_U8_8x8(mlib_u8 *curr_block,  
const mlib_u8 *ref_block, mlib_s32 stride);

mlib_status mlib_VideoCopyRef_U8_U8_8x4(mlib_u8 *curr_block,  
const mlib_u8 *ref_block, mlib_s32 stride);

Description  Each of these functions copies a block from the reference block to the current block. The stride  
applies to both the input reference block and the current block.

Parameters  Each of the functions takes the following arguments:

- **curr_block**  Pointer to the current block. curr_block must be 8-byte aligned.
- **ref_block**  Pointer to the reference block.
- **stride**  Stride, in bytes, between adjacent rows in both the current block and the  
reference block. stride must be a multiple of eight.

Return Values  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),  
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),  
mlib_VideoCopyRefAve_U8_U8(3MLIB), mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),  
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),  
mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB)

mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
**Name**
mlib_VideoCopyRef_U8_U8 — copies a block from the reference block to the current block

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoCopyRef_U8_U8(mlib_u8 *curr_block,
    const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
    mlib_s32 stride);
```

**Description**
The `mlib_VideoCopyRef_U8_U8()` function copies a block from the reference block to the current block. The stride applies to both the input reference block and the current block.

**Parameters**
The function takes the following arguments:

- **curr_block** Pointer to the current block. `curr_block` must be 8-byte aligned.
- **ref_block** Pointer to the reference block.
- **width** Width of the blocks.
- **height** Height of the blocks.
- **stride** Stride, in bytes, between adjacent rows in both the current block and the reference block. `stride` must be a multiple of eight.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8(3MLIB), mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8_16x16(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoCopyRef_U8_U8(3MLIB)

mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
**Name**  
mlib_VideoDCT16x16_S16_S16 – forward Discrete Cosine Transform (DCT)

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_VideoDCT16x16_S16_S16(mlib_s16 coeffs[256],
    const mlib_s16 block[256]);
```

**Description**  
The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The source and destination buffer addresses must be 8-byte aligned.

**Parameters**  
The function takes the following arguments:
- `coeffs`  
  Pointer to the destination DCT coefficients. `coeffs` must be 8-byte aligned.
- `block`  
  Pointer to an 16x16 motion-compensated block that is the difference between the reference block and the current block. `block` must be 8-byte aligned.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB), mlib_VideoDeQuantize_S16(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB), mlib_VideoQuantize_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)
The `mlib_VideoDCT16x16_S16_S16_B10()` function computes the forward DCT for the destination DCT coefficients of data type `mlib_s16` and source block of data type `mlib_s16`. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel which can occupy 10 bits, is represented as a 16-bit data.

**Parameters**
- `coeffs` Pointer to the output DCT coefficients. Note that `coeffs` must be 8-byte aligned.
- `block` Pointer to a 16x16 motion-compensated block which is the difference between the reference block and the current block. Note that `block` must be 8-byte aligned.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoDCT2x2_S16_S16(3MLIB)`, `mlib_VideoDCT4x4_S16_S16(3MLIB)`,
- `mlib_VideoDCT8x8_S16_S16(3MLIB)`, `mlib_VideoDCT8x8_S16_S16_B12(3MLIB)`,
- `mlib_VideoDCT8x8_S16_S16_NA(3MLIB)`, `mlib_VideoDCT8x8_S16_U8(3MLIB)`,
- `mlib_VideoDCT8x8_S16_U8_NA(3MLIB)`, `mlib_VideoDCT16x16_S16_S16(3MLIB)`,
- `mlib_VideoDeQuantize_S16(3MLIB)`, `mlib_VideoDeQuantizeInit_S16(3MLIB)`,
- `mlib_VideoQuantize_S16(3MLIB)`, `mlib_VideoQuantizeInit_S16(3MLIB)`, attributes(5)
The `mlib_VideoDCT2x2_S16_S16()` function computes the forward DCT for the destination DCT coefficients of data type `mlib_s16` and a source block of data type `mlib_s16`. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The source and destination buffer addresses must be 8-byte aligned.

The function takes the following arguments:

- `coeffs` Pointer to the destination DCT coefficients. `coeffs` must be 8-byte aligned.
- `block` Pointer to an 2x2 motion-compensated block that is the difference between the reference block and the current block. `block` must be 8-byte aligned.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_S16_B6(3MLIB), mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_S16_B6(3MLIB), mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDeQuantize_S16(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB), mlib_VideoQuantize_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)`
Name  mlib_VideoDCT4x4_S16_S16 – forward Discrete Cosine Transform (DCT)

Synopsis  cc [ flag... ] file... -tmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDCT4x4_S16_S16(mlib_s16 coeffs[16],
          const mlib_s16 block[16]);

Description  The mlib_VideoDCT4x4_S16_S16() function computes the forward DCT for the destination DCT coefficients of data type mlib_s16 and a source block of data type mlib_s16. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The source and destination buffer addresses must be 8-byte aligned.

Parameters  The function takes the following arguments:

coeffs  Pointer to the destination DCT coefficients. coeffs must be 8-byte aligned.

block  Pointer to an 4x4 motion-compensated block that is the difference between the reference block and the current block. block must be 8-byte aligned.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB),
mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB),
mlib_VideoDCT8x8_S16_S16_NA(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB),
mlib_VideoDCT8x8_S16_U8_NA(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB),
mlib_VideoDCT16x16_S16_S16_B10(3MLIB), mlib_VideoDeQuantize_S16(3MLIB),
mlib_VideoDeQuantizeInit_S16(3MLIB), mlib_VideoQuantize_S16(3MLIB),
mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)
Name  mlib_VideoDCT8x8Quantize_S16_S16_B12 – forward Discrete Cosine Transform (DCT) and quantization

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDCT8x8Quantize_S16_S16_B12(
    mlib_s16 coeffs[64], const mlib_s16 block[64],
    const mlib_d64 dqtable[64]);

Description  The mlib_VideoDCT8x8Quantize_S16_S16_B12() function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of mlib_VideoDCT8x8_S16_S16_B12() and mlib_VideoQuantize_S16() for better performance. The source to the DCT routine can occupy up to 12 bits for each of its elements, i.e., should be within the range of [-2048, 2047].

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

Parameters  The function takes the following arguments:

coeffs  Pointer to the quantized DCT coefficients. Note that coeffs must be 8-byte aligned.

block  Pointer to an 8x8 block. Note that block must be 8-byte aligned.

dqtable  Pointer to the quantization table generated by mlib_VideoQuantizeInit_S16().

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoDCT8x8Quantize_S16_S16_B12_NA(3MLIB),
mlib_VideoDCT8x8Quantize_S16_U8(3MLIB),
mlib_VideoDCT8x8Quantize_S16_U8_NA(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB),
attributes(5)
The `mlib_VideoDCT8x8Quantize_S16_S16_B12_NA()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mlib_VideoDCT8x8_S16_S16_B12_NA()` and `mlib_VideoQuantize_S16()` for better performance. The source to the DCT routine can occupy up to 12 bits for each of its elements, i.e., should be within the range of [-2048, 2047].

This function requires no special address alignment.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

**Parameters**
The function takes the following arguments:

- `coeffs`  Pointer to the quantized DCT coefficients.
- `block`  Pointer to an 8x8 block.
- `dqtable`  Pointer to the quantization table generated by `mlib_VideoQuantizeInit_S16()`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoDCT8x8Quantize_S16_S16_B12(3MLIB)`
- `mlib_VideoDCT8x8Quantize_S16_U8(3MLIB)`
- `mlib_VideoDCT8x8Quantize_S16_U8_NA(3MLIB)`
- `mlib_VideoQuantizeInit_S16(3MLIB)`
- `attributes(5)`
**Name**
mlib_VideoDCT8x8Quantize_S16_U8 – forward Discrete Cosine Transform (DCT) and quantization

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDCT8x8Quantize_S16_U8(
    mlib_s16 coeffs[64],
    const mlib_u8 *block,
    const mlib_d64 dqtable[64],
    mlib_s32 stride);

**Description**
The `mlib_VideoDCT8x8Quantize_S16_U8()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mlib_VideoDCT8x8_S16_U8()` and `mlib_VideoQuantize_S16()` for better performance.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters**
The function takes the following arguments:

- **coeffs**
  Pointer to the quantized DCT coefficients. Note that `coeffs` must be 8-byte aligned.

- **block**
  Pointer to an 8x8 block. Note that `block` must be 8-byte aligned.

- **dqtable**
  Pointer to the quantization table generated by `mlib_VideoQuantizeInit_S16()`.

- **stride**
  Stride in bytes between adjacent rows in the block. Note that `stride` must be a multiple of eight.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoDCT8x8Quantize_S16_S16_B12(3MLIB)`,
`mlib_VideoDCT8x8Quantize_S16_S16_B12 NA(3MLIB)`,
`mlib_VideoDCT8x8Quantize_S16_U8 NA(3MLIB)`,
`mlib_VideoQuantizeInit_S16(3MLIB)`
The `mlib_VideoDCT8x8Quantize_S16_U8_NA()` function computes the forward DCT and then quantizes the DCT coefficients. It's a combination of `mlib_VideoDCT8x8_S16_U8_NA()` and `mlib_VideoQuantize_S16()` for better performance.

This function requires no special address alignment.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters**

The function takes the following arguments:

- `coeffs` Pointer to the quantized DCT coefficients.
- `block` Pointer to an 8x8 block.
- `dqtable` Pointer to the quantization table generated by `mlib_VideoQuantizeInit_S16()`.
- `stride` Stride in bytes between adjacent rows in the block.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoDCT8x8Quantize_S16_S16_B12(3MLIB)`,
- `mlib_VideoDCT8x8Quantize_S16_S16_B12_NA(3MLIB)`,
- `mlib_VideoDCT8x8Quantize_S16_U8(3MLIB)`,
- `mlib_VideoQuantizeInit_S16(3MLIB)`,
- `attributes(5)`
mlib_VideoDCT8x8_S16_S16_B10

**Name**  
mlib_VideoDCT8x8_S16_S16_B10, mlib_VideoDCT8x8_S16_S16 – forward Discrete Cosine Transform (DCT)

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDCT8x8_S16_S16_B10(
    mlib_s16 coeffs[64], const mlib_s16 block[64]);

mlib_status mlib_VideoDCT8x8_S16_S16(
    mlib_s16 coeffs[64], const mlib_s16 block[64]);

**Description**  
The mlib_VideoDCT8x8_S16_S16_B10() function computes the forward DCT for the destination DCT coefficients of data type mlib_s16 and a source block of data type mlib_s16. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum.

The source and destination buffer addresses must be 8-byte aligned.

Since mediaLib 2.5, mlib_VideoDCT8x8_S16_S16() has been renamed to mlib_VideoDCT8x8_S16_S16_B10(). Now mlib_VideoDCT8x8_S16_S16() is an alias of mlib_VideoDCT8x8_S16_S16_B10().

**Parameters**  
The function takes the following arguments:

- **coeffs**  
  Pointer to the destination DCT coefficients. coeffs must be 8-byte aligned.

- **block**  
  Pointer to an 8x8 motion-compensated block that is the difference between the reference block and the current block. block must be 8-byte aligned.

**Return Values**  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B10 NA(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB), mlib_VideoDCT8x8_S16_U8 NA(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB), attributes(5)
The `mlib_VideoDCT8x8_S16_S16_B10_NA()` function computes the forward DCT for the destination DCT coefficients of data type `mlib_s16` and a source block of data type `mlib_s16`. The input to the DCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum.

This function requires no special address alignment.

Since mediaLib 2.5, `mlib_VideoDCT8x8_S16_S16_B10_NA()` has been renamed to `mlib_VideoDCT8x8_S16_S16_B10()` and `mlib_VideoDCT8x8_S16_S16_NA()` has been renamed to `mlib_VideoDCT8x8_S16_S16_B10_NA()`. Now `mlib_VideoDCT8x8_S16_S16_NA()` is an alias of `mlib_VideoDCT8x8_S16_S16_B10_NA()`.

The function takes the following arguments:

- `coeffs` Pointer to the destination DCT coefficients.
- `block` Pointer to an 8x8 motion-compensated block that is the difference between the reference block and the current block.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB), mlib_VideoDCT8x8_S16_U8_NA(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB), attributes(5)
**Name** mlib_VideoDCT8x8_S16_S16_B12 – forward Discrete Cosine Transform (DCT)

**Synopsis**

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>
mlib_status mlib_VideoDCT8x8_S16_S16_B12(
    mlib_s16 *coeffs[64],
    const mlib_s16 *block[64]);
```

**Description**
This function computes the forward DCT for the destination DCT coefficients of data type mlib_s16 and source block of data type mlib_s16. The source to the DCT routine can occupy up to 12 bits for each of its elements.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision.

**Parameters**
The function takes the following arguments:

- **coeffs**  Pointer to the output DCT coefficients. Note that `coeffs` must be 8-byte aligned.
- **block**   Pointer to an 8x8 block. Note that `block` must be 8-byte aligned.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB),
mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDCT8x8_S16_S16_B10 NA(3MLIB),
mlib_VideoDCT8x8_S16_U8(3MLIB), mlib_VideoDCT8x8_S16_U8 NA(3MLIB),
mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB),
attributes(5)
mlib_VideoDCT8x8_S16_U8 – forward Discrete Cosine Transform (DCT)

Synopsis
c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDCT8x8_S16_U8(mlib_s16 coeffs[64],
    const mlib_u8 *block, mlib_s32 stride);

Description
The mlib_VideoDCT8x8_S16_U8() function computes the forward DCT for the destination
DCT coefficients of data type mlib_s16 and source block of data type mlib_u8. The stride
applies to the block that is part of the frame currently being encoded.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

Parameters
The function takes the following arguments:
coeffs Pointer to the destination DCT coefficients. Note that coeffs must be 8-byte
aligned.
block Pointer to an 8x8 block in the current frame. Note that block must be 8-byte
aligned.
stride Stride in bytes between adjacent rows in the block. Note that stride must be a
multiple of eight.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB),
mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDCT8x8_S16_S16_B10_NA(3MLIB),
mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_U8_NA(3MLIB),
mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB),
mlib_VideoDeQuantize_S16(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB),
mlib_VideoQuantize_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)
Name  mlib_VideoDCT8x8_S16_U8_NA – forward Discrete Cosine Transform (DCT)
Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>
          mlib_status mlib_VideoDCT8x8_S16_U8_NA(mlib_s16 *coeffs[64],
                         const mlib_u8 *block, mlib_s32 stride);
Description  The mlib_VideoDCT8x8_S16_U8_NA() function computes the forward DCT for the
            destination DCT coefficients of data type mlib_s16 and source block of data type mlib_u8. The
            stride applies to the block that is part of the frame currently being encoded.
            This function requires no special address alignment.
            This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.
Parameters  The function takes the following arguments:
            coeffs   Pointer to the destination DCT coefficients.
            block    Pointer to an 8x8 block in the current frame.
            stride   Stride in bytes between adjacent rows in the block.
Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.
Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB),
           mlib_VideoDCT8x8_S16_S16_B10(3MLIB), mlib_VideoDCT8x8_S16_S16_B10 NA(3MLIB),
           mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB),
           mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB),
           mlib_VideoDeQuantize_S16(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB),
           mlib_VideoQuantize_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)
**mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12**

**Name**
mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12 – dequantization and inverse Discrete Cosine Transform (IDCT)

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12(
    mlib_s16 block[64], const mlib_s16 coeffs[64],
    const mlib_d64 dqtable[64]);
```

**Description**
The `mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mlib_VideoDeQuantize_S16()` and `mlib_VideoIDCT8x8_S16_S16_B12()` for better performance. The output of this function should be within the range of [-2048, 2047] if `coeffs` is obtained from the corresponding `mlib_VideoDCT8x8Quantize_S16_S16_B12()` function.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

**Parameters**
The function takes the following arguments:
- `block` Pointer to an 8x8 block. Note that `block` must be 8-byte aligned.
- `coeffs` Pointer to the input quantized DCT coefficients. Note that `coeffs` must be 8-byte aligned.
- `dqtable` Pointer to the dequantization table generated by `mlib_VideoDeQuantizeInit_S16()`.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12_NA(3MLIB)`,
- `mlib_VideoDeQuantizeIDCT8x8_U8_S16(3MLIB)`,
- `mlib_VideoDeQuantizeIDCT8x8_U8_S16_NA(3MLIB)`,
- `mlib_VideoDeQuantizeInit_S16(3MLIB)`, attributes(5)
The `mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12_NA()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mlib_VideoDeQuantize_S16()` and `mlib_VideoIDCT8x8_S16_S16_B12_NA()` for better performance. The output of this function should be within the range of [-2048, 2047] if `coeffs` is obtained from the corresponding `mlib_VideoIDCT8x8Quantize_S16_S16_B12_NA()` function.

This function requires no special address alignment.

This function can be used in JPEG with 12-bit precision, or in MPEG for the inter mode.

The function takes the following arguments:

- **block** Pointer to an 8x8 block.
- **coeffs** Pointer to the input quantized DCT coefficients.
- **dqtable** Pointer to the dequantization table generated by `mlib_VideoDeQuantizeInit_S16()`.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also `mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12(3MLIB)`, `mlib_VideoDeQuantizeIDCT8x8_U8_S16(3MLIB)`, `mlib_VideoDeQuantizeIDCT8x8_U8_S16_NA(3MLIB)`, `mlib_VideoDeQuantizeInit_S16(3MLIB)`, `attributes(5)`
The `mlib_VideoDe QuantizeIDCT8x8_U8_S16()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mlib_VideoDe Quantize_S16()` and `mlib_VideoIDCT8x8_U8_S16()` for better performance.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

**Parameters**

- **block**  
  Pointer to an 8x8 block. Note that `block` must be 8-byte aligned.

- **coeffs**  
  Pointer to the input quantized DCT coefficients. Note that `coeffs` must be 8-byte aligned.

- **dqtable**  
  Pointer to the dequantization table generated by `mlib_VideoDe QuantizeInit_S16()`.

- **stride**  
  Stride in bytes between adjacent rows in the block. Note that `stride` must be a multiple of eight.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `mlib_VideoDe QuantizeIDCT8x8_S16_S16_B12(3MLIB)`
- `mlib_VideoDe QuantizeIDCT8x8_S16_S16_B12 NA(3MLIB)`
- `mlib_VideoDe QuantizeIDCT8x8_U8_S16 NA(3MLIB)`
- `mlib_VideoDe QuantizeInit_S16(3MLIB)`, attributes(5)
### Name

`mlib_VideoDeQuantizeIDCT8x8_U8_S16_NA` - dequantization and inverse Discrete Cosine Transform (IDCT)

### Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDeQuantizeIDCT8x8_U8_S16_NA(
    mlib_u8 *block,
    const mlib_s16 coeffs[64],
    const mlib_d64 dqtable[64],
    mlib_s32 stride);
```

### Description

The `mlib_VideoDeQuantizeIDCT8x8_U8_S16_NA()` function dequantizes the quantized DCT coefficients and then computes the inverse DCT. It's a combination of `mlib_VideoDeQuantize_S16()` and `mlib_VideoIDCT8x8_U8_S16_NA()` for better performance.

This function requires no special address alignment.

This function can be used in JPEG with 8-bit precision, or in MPEG for the intra mode.

### Parameters

The function takes the following arguments:

- **block**  
  Pointer to an 8x8 block.
- **coeffs**  
  Pointer to the input quantized DCT coefficients.
- **dqtable**  
  Pointer to the dequantization table generated by `mlib_VideoDeQuantizeInit_S16()`.
- **stride**  
  Stride in bytes between adjacent rows in the block.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12(3MLIB)`,
- `mlib_VideoDeQuantizeIDCT8x8_S16_S16_B12_NA(3MLIB)`,
- `mlib_VideoDeQuantizeIDCT8x8_U8_S16(3MLIB)`,
- `mlib_VideoDeQuantizeInit_S16(3MLIB), attributes(5)`
mlib_VideoDeQuantizeInit_S16 – dequantization of forward Discrete Cosine Transform (DCT) coefficients

Synopsis
```
#include <mlib.h>

mlib_status mlib_VideoDeQuantizeInit_S16(mlib_d64 dqtable[64],
                           const mlib_s16 iqtable[64]);
```

Description
The `mlib_VideoDeQuantizeInit_S16()` function initializes the dequantization table.

The following equation is used:
\[
dqtable[i] = iqtable[i]; \quad 0 \leq i < 64
\]

Parameters
The function takes the following arguments:
- `dqtable` Pointer to dequantizer table coefficients.
- `iqtable` Pointer to original quantizer table coefficients:
  \[
  0 < iqtable[i] < 128
  \]

Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB),
mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB),
mlib_VideoDCT8x8_S16_S16_NA(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB),
mlib_VideoDCT8x8_S16_U8_NA(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB),
mlib_VideoDCT16x16_S16_S16_B10(3MLIB), mlib_VideoDeQuantize_S16(3MLIB),
mlib_VideoQuantize_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)
mlib_VideoDeQuantize_S16 – dequantization of forward Discrete Cosine Transform (DCT) coefficients

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDeQuantize_S16(mlib_s16 icoeffs[64],
const mlib_d64 dqtable[64]);

Description

The `mlib_VideoDeQuantize_S16()` function performs dequantization on DCT coefficients.

The following equation is used:

\[ icoeffs[i] = icoeffs[i] \times dqtable[i]; \quad 0 \leq i < 64 \]

Parameters

The function takes the following arguments:

- `icoeffs` Pointer to the output DCT coefficients:
  - \(-2048 < icoeffs[i] < 2048\)
  - Note that `icoeffs` must be 8-byte aligned.

- `dqtable` Pointer to dequantizer table coefficients.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_S16_NA(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB), mlib_VideoDCT8x8_S16_U8_NA(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_S16_B10(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB), mlib_VideoQuantize_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)`
mlib_VideoDownSample420 – down sampling rate conversion in JFIF

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDownSample420(mlib_u8 *dst, const mlib_u8 *src0,
const mlib_u8 *src1, mlib_s32 n);

Description
The mlib_VideoDownSample420() function performs down sampling rate conversion used in
JPEG File Interchange Format (JFIF).

Parameters
The function takes the following arguments:

dst Pointer to destination row. dst must be 8-byte aligned.

src0 Pointer to upper source row. src0 must be 8-byte aligned.

src1 Pointer to middle source row. src1 must be 8-byte aligned.

n Length of source rows. n must be greater than 1.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoDownSample420_S16(3MLIB), mlib_VideoDownSample422(3MLIB),
mlib_VideoDownSample422_S16(3MLIB), mlib_VideoUpSample420(3MLIB),
mlib_VideoUpSample420_Nearest(3MLIB),
mlib_VideoUpSample420_Nearest_S16(3MLIB), mlib_VideoUpSample420_S16(3MLIB),
mlib_VideoUpSample422(3MLIB), mlib_VideoUpSample422_S16(3MLIB),
mlib_VideoUpSample422_Nearest(3MLIB),
mlib_VideoUpSample422_Nearest_S16(3MLIB), attributes(5)
mlib_VideoDownSample420_S16(3MLIB)

**Name**
mlib_VideoDownSample420_S16 – down sampling rate conversion in JFIF

**Synopsis**
```c
cc [...] file [...] -mlib [...]
#include <mlib.h>
mlib_status mlib_VideoDownSample420_S16(mlib_s16 *dst,
const mlib_s16 *src0, const mlib_s16 *src1, mlib_s32 n);
```

**Description**
The `mlib_VideoDownSample420_S16()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters**
The function takes the following arguments:
- `dst` - Pointer to destination row. `dst` must be 8-byte aligned.
- `src0` - Pointer to upper source row. `src0` must be 8-byte aligned.
- `src1` - Pointer to middle source row. `src1` must be 8-byte aligned.
- `n` - Length of source rows. `n` must be greater than 1.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoDownSample420(3MLIB)`, `mlib_VideoDownSample422(3MLIB)`
- `mlib_VideoDownSample422_S16(3MLIB)`, `mlib_VideoUpSample420(3MLIB)`
- `mlib_VideoUpSample420_Nearest(3MLIB)`
- `mlib_VideoUpSample420_Nearest_S16(3MLIB)`
- `mlib_VideoUpSample422(3MLIB)`
- `mlib_VideoUpSample422_S16(3MLIB)`
- `mlib_VideoUpSample422_Nearest(3MLIB)`
- `mlib_VideoUpSample422_Nearest_S16(3MLIB)`

attributes(5)
# mlib_VideoDownSample422

## Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoDownSample422(mlib_u8 *dst, const mlib_u8 *src,
mlib_s32 n);
```

## Description

The `mlib_VideoDownSample422()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

## Parameters

The function takes the following arguments:

- `dst`: Pointer to destination row. `dst` must be 8-byte aligned.
- `src`: Pointer to source row. `src` must be 8-byte aligned.
- `n`: Length of source rows. `n` must be greater than 1.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_VideoDownSample420(3MLIB), mlib_VideoDownSample420_S16(3MLIB), mlib_VideoDownSample422_S16(3MLIB), mlib_VideoUpSample420(3MLIB), mlib_VideoUpSample420_Nearest(3MLIB), mlib_VideoUpSample420_Nearest_S16(3MLIB), mlib_VideoUpSample422(3MLIB), mlib_VideoUpSample422_S16(3MLIB), mlib_VideoUpSample422_Nearest(3MLIB), mlib_VideoUpSample422_Nearest_S16(3MLIB)`
The `mlib_VideoDownSample422_S16()` function performs down sampling rate conversion used in JPEG File Interchange Format (JFIF).

### Description
The `mlib_VideoDownSample422_S16()` function takes the following arguments:
- `dst`: Pointer to destination row. `dst` must be 8-byte aligned.
- `src`: Pointer to source row. `src` must be 8-byte aligned.
- `n`: Length of source rows. `n` must be greater than 1.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- `mlib_VideoDownSample420(3MLIB)`, `mlib_VideoDownSample420_S16(3MLIB)`, `mlib_VideoDownSample422(3MLIB)`, `mlib_VideoUpSample420(3MLIB)`, `mlib_VideoUpSample420_Nearest(3MLIB)`, `mlib_VideoUpSample420_S16(3MLIB)`, `mlib_VideoUpSample422(3MLIB)`, `mlib_VideoUpSample422_Nearest(3MLIB)`, `mlib_VideoUpSample422_S16(3MLIB)`, `attributes(5)`
The `mlib_VideoH263OverlappedMC_S16_U8()` function generates an 8x8 luminance prediction block (motion-compensated block) in the Advanced Prediction Mode for H.263 codec. The reference frame in this function is an interpolated frame. The output of this function must be added to the IDCT output in order to reconstruct the block in the current frame.

The following equation is used:

\[
\begin{align*}
    \text{mc}(x, y) &= \frac{(\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \cdot \text{H0}(x, y) + \text{ref}(2x + \text{mah}, 2y + \text{mav}) \cdot \text{H1}(x, y) + \text{ref}(2x + \text{mlh}, 2y + \text{mlv}) \cdot \text{H2}(x, y))}{8}; \\
    \text{for } x &= 0, 1, 2, 3; y = 0, 1, 2, 3 \\
    \text{mc}(x, y) &= \frac{(\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \cdot \text{H0}(x, y) + \text{ref}(2x + \text{mah}, 2y + \text{mav}) \cdot \text{H1}(x, y) + \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) \cdot \text{H2}(x, y))}{8}; \\
    \text{for } x &= 4, 5, 6, 7; y = 0, 1, 2, 3 \\
    \text{mc}(x, y) &= \frac{(\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \cdot \text{H0}(x, y) + \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) \cdot \text{H1}(x, y) + \text{ref}(2x + \text{mlh}, 2y + \text{mlv}) \cdot \text{H2}(x, y))}{8}; \\
    \text{for } x &= 0, 1, 2, 3; y = 4, 5, 6, 7 \\
    \text{mc}(x, y) &= \frac{(\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \cdot \text{H0}(x, y) + \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) \cdot \text{H1}(x, y) + \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) \cdot \text{H2}(x, y))}{8}; \\
    \text{for } x &= 4, 5, 6, 7; y = 4, 5, 6, 7 \\
    \text{mc}(x, y) &= \frac{(\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \cdot \text{H0}(x, y) + \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) \cdot \text{H1}(x, y) + \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) \cdot \text{H2}(x, y))}{8};
\end{align*}
\]

where

\[
\begin{align*}
    \text{H0} &= \begin{bmatrix} 4 & 5 & 5 & 5 & 5 & 5 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix} \\
    \text{H} &= \begin{bmatrix} 5 & 5 & 6 & 6 & 6 & 5 & 5 \\ 5 & 6 & 6 & 6 & 6 & 5 & 5 \\ 5 & 6 & 6 & 6 & 6 & 5 & 5 \end{bmatrix}
\end{align*}
\]
The function takes the following arguments:

- **mc_block** Pointer to the motion-compensated block.
- **ref_frame** Pointer to the interpolated reference frame.
- **mch** Horizontal coordinate of the motion vector for the current block.
- **mcv** Vertical coordinate of the motion vector for the current block.
- **mah** Horizontal coordinate of the motion vector for the block above the current block.
- **mav** Vertical coordinate of the motion vector for the block above the current block.
- **mbh** Horizontal coordinate of the motion vector for the block below the current block.
- **mbv** Vertical coordinate of the motion vector for the block below the current block.
- **mlh** Horizontal coordinate of the motion vector for the block to the left of the current block.
- **mlv** Vertical coordinate of the motion vector for the block to the left of the current block.
- **mrh** Horizontal coordinate of the motion vector for the block to the right of the current block.
- **mrv** Vertical coordinate of the motion vector for the block to the right of the current block.
ref_stride     Stride, in bytes, between adjacent rows in the interpolated reference frame.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes      See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
**Name**  
mlib_VideoH263OverlappedMC_U8_U8 – generates the 8x8 luminance prediction block in the Advanced Prediction Mode for H.263 codec

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>  
mlib_status mlib_VideoH263OverlappedMC_U8_U8(mlib_u8 *curr_block,  
  const mlib_u8 *ref_frame, mlib_s32 mch, mlib_s32 mcv, mlib_s32 mah,  
  mlib_s32 mav, mlib_s32 mbh, mlib_s32 mbv, mlib_s32 mlh, mlib_s32 mlv,  
  mlib_s32 mrh, mlib_s32 mrv, mlib_s32 curr_stride,  
  mlib_s32 ref_stride);

**Description**  
The `mlib_VideoH263OverlappedMC_U8_U8()` function generates an 8x8 luminance prediction block (motion-compensated block) in the Advanced Prediction Mode for H.263 codec. The reference frame in this function is an interpolated frame.

The following equation is used:

for x = 0, 1, 2, 3; y = 0, 1, 2, 3

\[
\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \ast \text{H0}(x, y) + \text{ref}(2x + \text{mah}, 2y + \text{mav}) \ast \text{H1}(x, y) + \text{ref}(2x + \text{mlh}, 2y + \text{mlv}) \ast \text{H2}(x, y)) / 8;
\]

for x = 4, 5, 6, 7; y = 0, 1, 2, 3

\[
\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \ast \text{H0}(x, y) + \text{ref}(2x + \text{mah}, 2y + \text{mav}) \ast \text{H1}(x, y) + \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) \ast \text{H2}(x, y)) / 8;
\]

for x = 0, 1, 2, 3; y = 4, 5, 6, 7

\[
\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \ast \text{H0}(x, y) + \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) \ast \text{H1}(x, y) + \text{ref}(2x + \text{mlh}, 2y + \text{mlv}) \ast \text{H2}(x, y)) / 8;
\]

for x = 4, 5, 6, 7; y = 4, 5, 6, 7

\[
\text{curr}(x, y) = (\text{ref}(2x + \text{mch}, 2y + \text{mcv}) \ast \text{H0}(x, y) + \text{ref}(2x + \text{mbh}, 2y + \text{mbv}) \ast \text{H1}(x, y) + \text{ref}(2x + \text{mrh}, 2y + \text{mrv}) \ast \text{H2}(x, y)) / 8;
\]

where

\[
\begin{align*}
\text{H0} &= \begin{bmatrix}
4 & 5 & 5 & 5 & 5 & 5 & 4 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 6 & 6 & 6 & 5 & 5 \\
\end{bmatrix} \\
\text{H1} &= \begin{bmatrix}
5 & 5 & 6 & 6 & 6 & 5 & 5 \\
5 & 5 & 6 & 6 & 6 & 5 & 5 \\
5 & 5 & 6 & 6 & 6 & 5 & 5 \\
\end{bmatrix} \\
\text{H2} &= \begin{bmatrix}
5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 \\
\end{bmatrix}
\end{align*}
\]
The function takes the following arguments:

- **curr_block**: Pointer to the current block.
- **ref_frame**: Pointer to the interpolated reference frame.
- **mch**: Horizontal coordinate of the motion vector for the current block.
- **mcv**: Vertical coordinate of the motion vector for the current block.
- **mah**: Horizontal coordinate of the motion vector for the block above the current block.
- **mav**: Vertical coordinate of the motion vector for the block above the current block.
- **mbh**: Horizontal coordinate of the motion vector for the block below the current block.
- **mbv**: Vertical coordinate of the motion vector for the block below the current block.
- **mlh**: Horizontal coordinate of the motion vector for the block to the left of the current block.
- **mlv**: Vertical coordinate of the motion vector for the block to the left of the current block.
- **mrh**: Horizontal coordinate of the motion vector for the block to the right of the current block.
- **mrv**: Vertical coordinate of the motion vector for the block to the right of the current block.
curr_stride Stride, in bytes, between adjacent rows in the current frame.
ref_stride Stride, in bytes, between adjacent rows in the interpolated reference frame.

Return Values The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
The `mlib_VideoIDCT8x8_S16_S16_B12()` function computes the inverse DCT (called IDCT) for the output IDCT block of data type `mlib_s16` and input DCT coefficients of data type `mlib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine should be within the range of \([-2048, 2047]\) if `coeffs` is obtained from the corresponding forward DCT function `mlib_VideoDCT8x8_S16_S16_B12()`.

The source and destination buffer addresses must be 8-byte aligned.

This function can be used in JPEG with 12-bit precision.

For MPEG, the output, which is really the difference between the current block and the reference block, can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

Since mediaLib 2.5, `mlib_VideoIDCT8x8_S16_S16()` has been renamed to `mlib_VideoIDCT8x8_S16_S16_B12()`. Now `mlib_VideoIDCT8x8_S16_S16()` is an alias of `mlib_VideoIDCT8x8_S16_S16_B12()`.

### Parameters
The function takes the following arguments:

- `block` Pointer to an 8x8 block in the current frame or motion-compensated reference block. `block` must be 8-byte aligned.
- `coeffs` Pointer to the source DCT coefficients. `coeffs` must be 8-byte aligned.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VideoIDCT_IEEE_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16_B12_NA(3MLIB),
mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB),
mlib_VideoIDCT8x8_S16_S16_Q1_Mismatch(3MLIB), mlib_VideoIDCT8x8_U8_S16(3MLIB),
mlib_VideoIDCT8x8_U8_S16_DC(3MLIB), mlib_VideoIDCT8x8_U8_S16_NA(3MLIB),
mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB), attributes(5)
mlib_VideoIDCT8x8_S16_S16_B12_NA

**Name**
mlib_VideoIDCT8x8_S16_S16_B12_NA, mlib_VideoIDCT8x8_S16_S16_NA – inverse Discrete Cosine Transform

**Synopsis**
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

```c
mlib_status mlib_VideoIDCT8x8_S16_S16_B12_NA(
    mlib_s16 block[64], const mlib_s16 coeffs[64]);
mlib_status mlib_VideoIDCT8x8_S16_S16_NA(
    mlib_s16 block[64], const mlib_s16 coeffs[64]);
```

**Description**
The `mlib_VideoIDCT8x8_S16_S16_B12_NA()` function computes the inverse DCT (called IDCT) for the output IDCT block of data type `mlib_s16` and input DCT coefficients of data type `mlib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine should be within the range of [-2048, 2047] if `coeffs` is obtained from the corresponding forward DCT function `mlib_VideoDCT8x8_S16_S16_B12_NA()`.

This function requires no special address alignment.

This function can be used in JPEG with 12-bit precision.

For MPEG, the output, which is really the difference between the current block and the reference block, can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

Since mediaLib 2.5, `mlib_VideoIDCT8x8_S16_S16_NA()` has been renamed to `mlib_VideoIDCT8x8_S16_S16_B12_NA()`. Now `mlib_VideoIDCT8x8_S16_S16_NA()` is an alias of `mlib_VideoIDCT8x8_S16_S16_B12_NA()`.

**Parameters**
The function takes the following arguments:

- `block` Pointer to an 8x8 block in the current frame or motion-compensated reference block. `block` need not be 8-byte aligned.
- `coeffs` Pointer to the source DCT coefficients. `coeffs` need not be 8-byte aligned.

**Return Values**
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VideoIDCT_IEEE_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16_B12(3MLIB),
mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB),
mlib_VideoIDCT8x8_S16_S16_Q1_Mismatch(3MLIB), mlib_VideoIDCT8x8_U8_S16(3MLIB),
mlib_VideoIDCT8x8_U8_S16_DC(3MLIB), mlib_VideoIDCT8x8_U8_S16_NA(3MLIB),
mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB), attributes(5)
# mlib_VideoIDCT8x8_S16_S16_DC

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_VideoIDCT8x8_S16_S16_DC(mlib_s16 *block,
                                           const mlib_s16 *coeffs);
```

## Description

The `mlib_VideoIDCT8x8_S16_S16_DC()` function can be used only when \( F(0,0) \) is nonzero. It computes the inverse DCT (called IDCT) for the output IDCT block of data type `mlib_s16` and input DCT coefficients of data type `mlib_s16`. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

## Parameters

The function takes the following arguments:

- `block` Pointer to the current block. `block` must be 8-byte aligned.
- `coeffs` Pointer to the source DCT coefficients. `coeffs` must be 8-byte aligned.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

`mlib_VideoIDCT_IEEE_S16_S16(3MLIB)`, `mlib_VideoIDCT8x8_S16_S16(3MLIB)`, `mlib_VideoIDCT8x8_S16_S16_NA(3MLIB)`, `mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16_DC(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16_NA(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB)`, `attributes(5)`
Name  
mllib_VideoIDCT8x8_S16_S16_Q1 – inverse Discrete Cosine Transform

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoIDCT8x8_S16_S16_Q1(mlib_s16 *block[64],
    const mlib_s16 *coeffs[64]);

Description  
The mlib_VideoIDCT8x8_S16_S16_Q1() function can be used only when \( F(u,v) \) are nonzero for \( 0 \leq u < 4 \) and \( 0 \leq v < 4 \).

Parameters  
The function takes the following arguments:

- **block**: Pointer to the current block. block must be 8-byte aligned.
- **coeffs**: Pointer to the source DCT coefficients. coeffs must be 8-byte aligned.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoIDCT_IEEE_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16(3MLIB),
mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_NA(3MLIB),
mlib_VideoIDCT8x8_U8_S16(3MLIB), mlib_VideoIDCT8x8_U8_S16_DC(3MLIB),
mlib_VideoIDCT8x8_U8_S16_NA(3MLIB), mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB),
attributes(5)
**Name**
milib_VideoIDCT8x8_S16_S16_Q1_Mismatch – inverse Discrete Cosine Transform

**Synopsis**
```
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoIDCT8x8_S16_S16_Q1_Mismatch(
    mlib_s16 block[64], const mlib_s16 coeffs[64]);
```

**Description**
The `mlib_VideoIDCT8x8_S16_S16_Q1_Mismatch()` function computes the inverse IDCT in the inter mode.

This function is similar to `mlib_VideoIDCT8x8_S16_S16_Q1()` which should only be used when `coeffs[u][v] (u, v = 0...7)` are non-zero only for `u` and `v` less then 4. However, this function also allows element `coeffs[7][7]` to be non-zero. The primary benefit of this modification is that it can handle situations where `coeffs[7][7]` has been made non-zero by MPEG mismatch-control, allowing a simplified version of the IDCT to be undertaken for a much larger number of situations.

**Parameters**
The function takes the following arguments:

- `block` Pointer to an 8x8 motion-compensated block which is the difference between the reference block and current block. `block` must be 8-byte aligned.

- `coeffs` Pointer to the input DCT coefficients. `coeffs` must be 8-byte aligned. `coeffs` should be in S12 range or it should be obtained from the corresponding forward DCT.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB)`, attributes(5)
# mlib_VideoIDCT8x8_U8_S16

**Synopsis**

```c
#include <mlib.h>
mlib_status mlib_VideoIDCT8x8_U8_S16(mlib_u8 *block,
    const mlib_s16 coeffs[64], mlib_s32 stride);
```

**Description**

The `mlib_VideoIDCT8x8_U8_S16()` function computes the inverse DCT (called IDCT) for the destination IDCT block of data type `mlib_u8` and source DCT coefficients of data type `mlib_s16`.

The stride applies to the block that is part of the frame currently being reconstructed.

**Parameters**

The function takes the following arguments:

- `block` : Pointer to an 8x8 block in the current frame. Block must be 8-byte aligned.
- `coeffs` : Pointer to the source DCT coefficients. Coeffs must be 8-byte aligned.
- `stride` : Stride, in bytes, between adjacent rows in a block. Stride must be a multiple of eight.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_VideoIDCT_16x16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16(3MLIB),
mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_DC(3MLIB),
mlib_VideoIDCT8x8_S16_DC(3MLIB), mlib_VideoIDCT8x8_U8_S16_DC(3MLIB),
mlib_VideoIDCT8x8_U8_S16_DC(3MLIB), mlib_VideoIDCT8x8_U8_S16_DC(3MLIB),
attributes(5)
Name  mlib_VideoIDCT8x8_U8_S16_DC – inverse Discrete Cosine Transform

Synopsis  

cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_VideoIDCT8x8_U8_S16_DC(mlib_u8 *block,  
const mlib_s16 *coeffs, mlib_s32 stride);

Description  The mlib_VideoIDCT8x8_U8_S16_DC() function can be used only when \( F(0,0) \) is nonzero. It computes the inverse DCT (called IDCT) for the destination IDCT block of data type mlib_u8 and source DCT coefficients of data type mlib_s16. This function is not guaranteed to be IEEE-1180-compliant. The output of the IDCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

Parameters  The function takes the following arguments:
- block  Pointer to the current block. block must be 8-byte aligned.
- coeffs  Pointer to the source DCT coefficients. coeffs must be 8-byte aligned.
- stride  Stride, in bytes, between adjacent rows in a block. stride must be a multiple of eight.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoIDCT_IEEE_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_NA(3MLIB), mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB), mlib_VideoIDCT8x8_U8_S16(3MLIB), mlib_VideoIDCT8x8_U8_S16_NA(3MLIB), mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB), attributes(5)
The `mlib_VideoIDCT8x8_U8_S16_NA()` function computes the inverse DCT (called IDCT) for the destination IDCT block of data type `mlib_u8` and source DCT coefficients of data type `mlib_s16`.

The `stride` applies to the block that is part of the frame currently being reconstructed.

This function requires no special address alignment.

### Parameters

- **block**
  - Pointer to the current block.
- **coeffs**
  - Pointer to the source DCT coefficients.
- **stride**
  - Stride, in bytes, between adjacent rows in the block.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_VideoIDCT_IEEE_S16_S16(3MLIB)`, `mlib_VideoIDCT8x8_S16_S16(3MLIB)`, `mlib_VideoIDCT8x8_S16_S16_DC(3MLIB)`, `mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16_DC(3MLIB)`, `mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB)`
- `attributes(5)`
### Name
mlib_VideoIDCT8x8_U8_S16_Q1 – inverse Discrete Cosine Transform

### Synopsis
```c
#include <mlib.h>

mlib_status mlib_VideoIDCT8x8_U8_S16_Q1(mlib_u8 *block,
const mlib_s16 coeffs[64], mlib_s32 stride);
```

### Description
The `mlib_VideoIDCT8x8_U8_S16_Q1()` function can be used only when \( F(u, v) \) are nonzero and only when \( 0 \leq u < 4 \) and \( 0 \leq v < 4 \). The stride applies to the block that is part of the frame currently being reconstructed.

### Parameters
The function takes the following arguments:

- **block**  
  Pointer to the current block. block must be 8-byte aligned

- **coeffs**  
  Pointer to the source DCT coefficients. coeffs must be 8-byte aligned.

- **stride**  
  Stride, in bytes, between adjacent rows in a block. stride must be a multiple of eight.

### Return Values
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
`mlib_VideoIDCT_IEEE_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16(3MLIB),
mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_NA(3MLIB),
mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB), mlib_VideoIDCT8x8_U8_S16(3MLIB),
mlib_VideoIDCT8x8_U8_S16_DC(3MLIB), mlib_VideoIDCT8x8_U8_S16_NA(3MLIB),
attributes(5)`
The `mlib_VideoIDCT_IEEE_S16_S16()` function computes the inverse DCT (called IDCT) for the output IDCT block of data type `mlib_s16` and input DCT coefficients of data type `mlib_s16`. This function is guaranteed to be IEEE-1180-compliant. The output of the IDCT routine is the difference between the current block and the reference block. The difference pixel can occupy nine bits and is represented as a 16-bit datum. The output must be added to the motion-compensated reference block in order to reconstruct the current block.

The function takes the following arguments:

- `block`  
  Pointer to an 8x8 motion-compensated block that is the difference between the reference block and the current block. `block` need not be 8-byte aligned.

- `coeffs`  
  Pointer to the source DCT coefficients. `coeffs` need not be 8-byte aligned.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See also `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
`mlib_VideoIDCT8x8_S16_S16(3MLIB), mlib_VideoIDCT8x8_S16_S16_DC(3MLIB), mlib_VideoIDCT8x8_S16_S16_NA(3MLIB), mlib_VideoIDCT8x8_S16_S16_Q1(3MLIB), mlib_VideoIDCT8x8_U8_S16(3MLIB), mlib_VideoIDCT8x8_U8_S16_DC(3MLIB), mlib_VideoIDCT8x8_U8_S16_NA(3MLIB), mlib_VideoIDCT8x8_U8_S16_Q1(3MLIB), attributes(5)`
Each of these functions performs half-pixel interpolation in the X direction and averaging for a reference block of data type mlib_u8 and a current block of data type mlib_u8. The stride applies to both the input reference block and the current block.

Parameters:
- `curr_block`: Pointer to the current block. `curr_block` must be 8-byte aligned.
- `ref_block`: Pointer to the reference block.
- `frm_stride`: Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. `frm_stride` must be a multiple of eight.
- `fld_stride`: Stride, in bytes, between adjacent rows in a field in both the current block and reference block. `fld_stride` must be a multiple of eight.

Return Values:
Each of the functions returns MLIB_SUCCESS if successful. Otherwise, it returns MLIB_FAILURE.

Attributes:
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also

- `mlib_VideoAddBlock_U8_S16(3MLIB)`, `mlib_VideoCopyRef_S16_U8(3MLIB)`, `mlib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mlib_VideoCopyRef_U8_U8_16x16(3MLIB)`, `mlib_VideoCopyRefAve_U8_U8(3MLIB)`, `mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`, `mlib_VideoCopyRefAveX_U8_U8_16x16(3MLIB)`, `mlib_VideoCopyRefAveY_U8_U8(3MLIB)`, `mlib_VideoCopyRefAveY_U8_U8_16x16(3MLIB)`, `mlib_VideoH263OverlappedMC_S16_U8(3MLIB)`, `mlib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mlib_VideoH263OverlappedMC_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpAveX_U8_U8(3MLIB)`, `mlib_VideoInterpAveXY_U8_U8(3MLIB)`, `mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpAveY_U8_U8(3MLIB)`, `mlib_VideoInterpAveY_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpX_S16_U8(3MLIB)`, `mlib_VideoInterpX_S16_U8_16x16(3MLIB)`, `mlib_VideoInterpX_U8_U8(3MLIB)`, `mlib_VideoInterpX_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpXY_S16_U8(3MLIB)`, `mlib_VideoInterpXY_S16_U8_16x16(3MLIB)`, `mlib_VideoInterpXY_U8_U8(3MLIB)`, `mlib_VideoInterpXY_U8_U8_16x16(3MLIB)`, `mlib_VideoInterpY_S16_U8(3MLIB)`, `mlib_VideoInterpY_S16_U8_16x16(3MLIB)`, `mlib_VideoInterpY_U8_U8(3MLIB)`, `mlib_VideoInterpY_U8_U8_16x16(3MLIB)`, `mlib_VideoP64Decimate_U8_U8(3MLIB)`, `mlib_VideoP64Decimate_U8_U8_16x16(3MLIB)`, `mlib_VideoP64DecimateU8_U8(3MLIB)`, `mlib_VideoP64DecimateU8_U8_16x16(3MLIB)`, `mlib_VideoP64Loop_S16_U8(3MLIB)`, `mlib_VideoP64Loop_S16_U8_16x16(3MLIB)`, `mlib_VideoP64Loop_U8_U8(3MLIB)`, `mlib_VideoP64Loop_U8_U8_16x16(3MLIB)`, `attributes(5)`
mlib_VideoInterpAveX_U8_U8(3MLIB)

Name
mlib_VideoInterpAveX_U8_U8 – half-pixel interpolation in the X direction and averaging for reference block

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpAveX_U8_U8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
mlib_s32 frm_stride, mlib_s32 fld_stride);

Description
The mlib_VideoInterpAveX_U8_U8() function performs half-pixel interpolation in the X direction and averaging for a reference block of data type mlib_u8 and a current block of data type mlib_u8. The stride applies to both the input reference block and the current block.

Parameters
The function takes the following arguments:

curr_block    Pointer to the current block. curr_block must be 8-byte aligned.

ref_block    Pointer to the reference block.

width    Width of the blocks.

height    Height of the blocks.

frm_stride    Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm_stride must be a multiple of eight.

fld_stride    Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16, mlib_VideoInterpAveXY_U8_U8_16x8, mlib_VideoInterpAveXY_U8_U8_8x16, mlib_VideoInterpAveXY_U8_U8_8x8, mlib_VideoInterpAveXY_U8_U8_8x4 – half-pixel interpolation in the X and Y directions and averaging for reference block

#include <mlib.h>

mlib_status mlib_VideoInterpAveXY_U8_U8_16x16(mlib_u8 *curr_block,
                           const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveXY_U8_U8_16x8(mlib_u8 *curr_block,
                           const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveXY_U8_U8_8x16(mlib_u8 *curr_block,
                           const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveXY_U8_U8_8x8(mlib_u8 *curr_block,
                           const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveXY_U8_U8_8x4(mlib_u8 *curr_block,
                           const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

Each of these functions performs half-pixel interpolation in the X and Y directions and averaging for a reference block of data. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Description

Parameters

Each of the functions takes the following arguments:

curr_block Pointer to the current block. curr_block must be 8-byte aligned.

ref_block Pointer to the reference block.

frm_stride Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm_stride must be a multiple of eight.

fld_stride Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Name  mlib_VideoInterpAveXY_U8_U8 – half-pixel interpolation in the X and Y directions and averaging for reference block

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpAveXY_U8_U8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
mlib_s32 frm_stride, mlib_s32 fld_stride);

Description  The mlib_VideoInterpAveXY_U8_U8() function performs half-pixel interpolation in the X and Y directions and averaging for a reference block of data type mlib_u8 and a current block of data type mlib_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Parameters  The function takes the following arguments:

curr_block  Pointer to the current block. curr_block must be 8-byte aligned.
ref_block  Pointer to the reference block.
width  Width of the blocks.
height  Height of the blocks.
frm_stride  Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm_stride must be a multiple of eight.
fld_stride  Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB), mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB), mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB), mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB), mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8(3MLIB)

mlib_VideoInterpX_S16_U8_U8(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_U8(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_U8(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_U8(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
mlib_VideoInterpAveY_U8_U8_16x16, mlib_VideoInterpAveY_U8_U8_16x8, mlib_VideoInterpAveY_U8_U8_8x16, mlib_VideoInterpAveY_U8_U8_8x8, mlib_VideoInterpAveY_U8_U8_8x4 – half-pixel interpolation in the Y direction and averaging for reference block

Synopsis

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpAveY_U8_U8_16x16(mlib_u8 *curr_block,
        const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveY_U8_U8_16x8(mlib_u8 *curr_block,
        const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveY_U8_U8_8x16(mlib_u8 *curr_block,
        const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveY_U8_U8_8x8(mlib_u8 *curr_block,
        const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

mlib_status mlib_VideoInterpAveY_U8_U8_8x4(mlib_u8 *curr_block,
        const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);

Description

Each of these functions performs half-pixel interpolation in the Y direction and averaging for a reference block of data type mlib_u8 and a current block of data type mlib_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Parameters

Each of the functions takes the following arguments:

- **curr_block**  Pointer to the current block. curr_block must be 8-byte aligned.
- **ref_block**  Pointer to the reference block.
- **frm_stride**  Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm_stride must be a multiple of eight.
- **fld_stride**  Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values

Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),  
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),  
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),  
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),  
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),  
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),  
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),  
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),  
mlib_VideoInterpX_S16_U8(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),  
mlib_VideoInterpX_U8_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),  
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),  
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),  
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),  
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Name  
mlib_VideoInterpAveY_U8_U8 – half-pixel interpolation in the Y direction and averaging for reference block

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpAveY_U8_U8(mlib_u8 *curr_block,
    const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
    mlib_s32 frm_stride, mlib_s32 fld_stride);

Description  
The mlib_VideoInterpAveY_U8_U8() function performs half-pixel interpolation in the Y direction and averaging for a reference block of data type mlib_u8 and a current block of data type mlib_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Parameters  
The function takes the following arguments:

curr_block  Pointer to the current block. curr_block must be 8-byte aligned.
ref_block  Pointer to the reference block.
width  Width of the blocks.
height  Height of the blocks.
frm_stride  Stride, in bytes, between adjacent rows in a frame in both the current block and the reference block. frm_stride must be a multiple of eight.
fld_stride  Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8(3MLIB)

mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Each of these functions performs half-pixel interpolation in the X direction for a reference block of data type mlib_u8 and a current block of data type mlib_s16. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

### Parameters

Each of the functions takes the following arguments:

- **mc_block**
  - Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.

- **ref_block**
  - Pointer to the reference block.

- **frm_stride**
  - Stride, in bytes, between adjacent rows in a frame in the reference block. frm_stride must be a multiple of eight.

- **fld_stride**
  - Stride, in bytes, between adjacent rows in a field in the reference block. fld_stride must be a multiple of eight.

### Return Values

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

### Attributes

See attributes(5) for descriptions of the following attributes:
### ATTRIBUTE TYPE | ATTRIBUTE VALUE
---|---
Interface Stability | Committed
MT-Level | MT-Safe

See Also

- `mlib_VideoAddBlock_U8_S16(3MLIB)`, `mlib_VideoCopyRef_S16_U8(3MLIB)`, `mlib_VideoCopyRef_S16_U8_U8(3MLIB)`, `mlib_VideoCopyRef_U8_U8_U8(3MLIB)`, `mlib_VideoCopyRefAve_U8_U8(3MLIB)`, `mlib_VideoH263OverlappedMC_S16_U8(3MLIB)`, `mlib_VideoH263OverlappedMC_U8_U8(3MLIB)`, `mlib_VideoInterpAveX_U8_U8_U8(3MLIB)`, `mlib_VideoInterpAveXY_U8_U8_U8(3MLIB)`, `mlib_VideoInterpAveY_U8_U8_U8(3MLIB)`, `mlib_VideoInterpX_S16_U8_U8(3MLIB)`, `mlib_VideoInterpX_SI6_U8_U8(3MLIB)`, `mlib_VideoInterpXY_U8_U8_U8(3MLIB)`, `mlib_VideoInterpY_S16_U8_U8(3MLIB)`, `mlib_VideoInterpY_SI6_U8_U8(3MLIB)`, `mlib_VideoInterpY_U8_U8_U8(3MLIB)`, `mlib_VideoP64Decimate_U8_U8(3MLIB)`, `mlib_VideoP64Loop_SI6_U8(3MLIB)`
# mlib_VideoInterpX_S16_U8

## Name
mlib_VideoInterpX_S16_U8 – half-pixel interpolation in the X direction

## Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpX_S16_U8(mlib_s16 *mc_block,
                    const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
                    mlib_s32 frm_stride, mlib_s32 fld_stride);
```

## Description

The `mlib_VideoInterpX_S16_U8()` function performs half-pixel interpolation in the X direction for a reference block of data type `mlib_u8` and a current block of data type `mlib_s16`. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

## Parameters

The function takes the following arguments:

- **mc_block**: Pointer to the motion-compensated reference block. `mc_block` must be 8-byte aligned.
- **ref_block**: Pointer to the reference block.
- **width**: Width of the blocks.
- **height**: Height of the blocks.
- **frm_stride**: Stride, in bytes, between adjacent rows in a frame in the reference block. `frm_stride` must be a multiple of eight.
- **fld_stride**: Stride, in bytes, between adjacent rows in a field in the reference block. `fld_stride` must be a multiple of eight.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_VideoAddBlock_U8_S16(3MLIB)`, `mlib_VideoCopyRef_S16_U8(3MLIB)`
- `mlib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mlib_VideoCopyRef_U8 UB(3MLIB)`,
- `mlib_VideoCopyRef_U8 UB 16x16(3MLIB)`, `mlib_VideoCopyRefAve_U8 UB(3MLIB)`
- `mlib_VideoCopyRefAve_U8 16x16(3MLIB)`, `mlib_VideoH263OverlappedMC_S16_U8(3MLIB)`,
- `mlib_VideoH263OverlappedMC UB UB(3MLIB)`, `mlib_VideoInterpAveX_U8 UB(3MLIB)`
- `mlib_VideoInterpAveX UB 16x16(3MLIB)`, `mlib_VideoInterpAveXY UB UB(3MLIB)`,
- `mlib_VideoInterpAveXY UB UB 16x16(3MLIB)`
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
mlib_VideoInterpX_U8_U8_16x16, mlib_VideoInterpX_U8_U8_16x8,
mlib_VideoInterpX_U8_U8_8x16, mlib_VideoInterpX_U8_U8_8x8,
mlib_VideoInterpX_U8_U8_8x4 – half-pixel interpolation in the X direction

#include <mlib.h>

mlib_status mlib_VideoInterpX_U8_U8_16x16(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpX_U8_U8_16x8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpX_U8_U8_8x16(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpX_U8_U8_8x8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpX_U8_U8_8x4(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

Description Each of these functions performs half-pixel interpolation in the X direction for a reference block of data type mlib_u8 and a current block of data type mlib_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Parameters Each of the functions takes the following arguments:

- **curr_block** Pointer to the current block. curr_block must be 8-byte aligned.
- **ref_block** Pointer to the reference block.
- **frm_stride** Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. frm_stride must be a multiple of eight.
- **fld_stride** Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
ATTRIBUTE TYPE | ATTRIBUTE VALUE
--- | ---
MT-Level | MT-Safe

See Also

- `mlib_VideoAddBlock_U8_S16(3MLIB)`
- `mlib_VideoCopyRef_S16_U8(3MLIB)`
- `mlib_VideoCopyRef_U8_S16(3MLIB)`
- `mlib_VideoCopyRef_U8_U8(3MLIB)`
- `mlib_VideoCopyRefAve_U8_U8(3MLIB)`
- `mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`
- `mlib_VideoCopyRefAveX_U8_U8(3MLIB)`
- `mlib_VideoCopyRefAveXY_U8_U8(3MLIB)`
- `mlib_VideoCopyRefAveY_U8_U8(3MLIB)`
- `mlib_VideoH263OverlappedMC_S16_U8(3MLIB)`
- `mlib_VideoInterpAveX_U8_U8(3MLIB)`
- `mlib_VideoInterpAveXY_U8_U8(3MLIB)`
- `mlib_VideoInterpAveY_U8_U8(3MLIB)`
- `mlib_VideoInterpX_S16_U8(3MLIB)`
- `mlib_VideoInterpXY_S16_U8(3MLIB)`
- `mlib_VideoInterpY_S16_U8(3MLIB)`
- `mlib_VideoP64Decimate_U8_U8(3MLIB)`
- `mlib_VideoP64Loop_S16_U8(3MLIB)`
- `mlib_VideoP64Loop_U8_U8(3MLIB)`
- `attributes(5)`
### Synopsis

```c
#include <mlib.h>

mlib_status mlib_VideoInterpX_U8_U8(mlib_u8 *curr_block,
    const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
    mlib_s32 frm_stride, mlib_s32 fld_stride);
```

### Description

The `mlib_VideoInterpX_U8_U8()` function performs half-pixel interpolation in the X direction for a reference block of data type `mlib_u8` and a current block of data type `mlib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

### Parameters

- `curr_block`: Pointer to the current block. `curr_block` must be 8-byte aligned.
- `ref_block`: Pointer to the reference block.
- `width`: Width of the blocks.
- `height`: Height of the blocks.
- `frm_stride`: Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. `frm_stride` must be a multiple of eight.
- `fld_stride`: Stride, in bytes, between adjacent rows in a field in both the current block and reference block. `fld_stride` must be a multiple of eight.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- mlib_VideoAddBlock_U8_S16(3MLIB)
- mlib_VideoCopyRef_S16_U8(3MLIB)
- mlib_VideoCopyRef_U8_U8_16x16(3MLIB)
- mlib_VideoCopyRef_U8_U8_16x16(3MLIB)
- mlib_VideoCopyRefU8_U8_16x16(3MLIB)
- mlib_VideoCopyRefU8_U8_16x16(3MLIB)
- mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)
- mlib_VideoCopyRefAveU8_U8_16x16(3MLIB)
- mlib_VideoH263OverlappedMC_S16_U8(3MLIB)
- mlib_VideoH263OverlappedMC_U8_U8(3MLIB)
- mlib_VideoInterpRefX_U8_U8_16x16(3MLIB)
- mlib_VideoInterpRefXY_U8_U8_16x16(3MLIB)
- mlib_VideoInterpRefXYU8_U8_16x16(3MLIB)
- mlib_VideoInterpX_U8_U8_16x16(3MLIB)
- mlib_VideoInterpX_S16_U8(3MLIB)
- mlib_VideoInterpX_S16_U8_16x16(3MLIB)
- mlib_VideoInterpXY_S16_U8(3MLIB)
- mlib_VideoInterpXY_S16_U8_16x16(3MLIB)
- mlib_VideoInterpXYU8_U8_16x16(3MLIB)
- mlib_VideoInterpXYU8_U8_16x16(3MLIB)
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Each of these functions performs half-pixel interpolation in the X and Y directions for a reference block of data type `mlib_u8` and a current block of data type `mlib_s16`. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

### Parameters
Each of the functions takes the following arguments:

- **mc_block**: Pointer to the motion-compensated reference block. `mc_block` must be 8-byte aligned.
- **ref_block**: Pointer to the reference block.
- **frm_stride**: Stride, in bytes, between adjacent rows in a frame in the reference block. `frm_stride` must be a multiple of eight.
- **fld_stride**: Stride, in bytes, between adjacent rows in a field in the reference block. `fld_stride` must be a multiple of eight.

### Return Values
Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

### Attributes
See attributes(5) for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),  
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),  
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),  
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),  
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),  
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),  
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),  
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),  
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),  
mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),  
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),  
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),  
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
The `mlib_VideoInterpXY_S16_U8()` function performs half-pixel interpolation in the X and Y directions for a reference block of data type `mlib_u8` and a current block of data type `mlib_s16`. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

The function takes the following arguments:

- `mc_block` Pointer to the motion-compensated reference block. `mc_block` must be 8-byte aligned.
- `ref_block` Pointer to the reference block.
- `width` Width of the blocks.
- `height` Height of the blocks.
- `frm_stride` Stride, in bytes, between adjacent rows in a frame in the reference block. `frm_stride` must be a multiple of eight.
- `fld_stride` Stride, in bytes, between adjacent rows in a field in the reference block. `fld_stride` must be a multiple of eight.

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoAddBlock_U8_S16(3MLIB)`
- `mlib_VideoCopyRef_S16_U8(3MLIB)`
- `mlib_VideoCopyRef_S16_U8_16x16(3MLIB)`
- `mlib_VideoCopyRef_U8_U8_16x16(3MLIB)`
- `mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`
- `mlib_VideoH263OverlappedMC_S16_U8(3MLIB)`
- `mlib_VideoH263OverlappedMC_U8_U8(3MLIB)`
- `mlib_VideoInterpAveX_U8_U8(3MLIB)`
- `mlib_VideoInterpAveX_U8_U8_16x16(3MLIB)`
- `mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB)`
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
mlib_VideoInterpXY_U8_U8_16x16, mlib_VideoInterpXY_U8_U8_16x8,
mlib_VideoInterpXY_U8_U8_8x16, mlib_VideoInterpXY_U8_U8_8x8,
mlib_VideoInterpXY_U8_U8_8x4 – half-pixel interpolation in the X and Y directions

Synopsis
#include <mlib.h>

mlib_status mlib_VideoInterpXY_U8_U8_16x16(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpXY_U8_U8_16x8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpXY_U8_U8_8x16(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpXY_U8_U8_8x8(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

mlib_status mlib_VideoInterpXY_U8_U8_8x4(mlib_u8 *curr_block,
const mlib_u8 *ref_block, mlib_s32 frm_stride,
mlib_s32 fld_stride);

Description
Each of these functions performs half-pixel interpolation in the X and Y directions for a
reference block of data type mlib_u8 and a current block of data type mlib_u8. In this mode,
the motion-compensated reference block becomes the current block. Thus, the stride applies
to both the input reference block and the current block.

Parameters
Each of the functions takes the following arguments:

- **curr_block**: Pointer to the current block. curr_block must be 8-byte aligned.
- **ref_block**: Pointer to the reference block.
- **frm_stride**: Stride, in bytes, between adjacent rows in a frame in both the current block and
  reference block. frm_stride must be a multiple of eight.
- **fld_stride**: Stride, in bytes, between adjacent rows in a field in both the current block and
  reference block. fld_stride must be a multiple of eight.

Return Values
Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE_TYPE</th>
<th>ATTRIBUTE_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>ATTRIBUTE TYPE</td>
<td>ATTRIBUTE VALUE</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OvelappedMC_S16_U8(3MLIB),
mlib_VideoH263OvelappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveXU_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpX_U8_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
**Name** mlib_VideoInterpXY_U8_U8 – half-pixel interpolation in the X and Y directions

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpXY_U8_U8(mlib_u8 *curr_block,
                           const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
                           mlib_s32 frm_stride, mlib_s32 fld_stride);
```

**Description** The `mlib_VideoInterpXY_U8_U8()` function performs half-pixel interpolation in the X and Y directions for a reference block of data type `mlib_u8` and a current block of data type `mlib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters** The function takes the following arguments:
- `curr_block` Pointer to the current block. `curr_block` must be 8-byte aligned.
- `ref_block` Pointer to the reference block.
- `width` Width of the blocks.
- `height` Height of the blocks.
- `frm_stride` Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. `frm_stride` must be a multiple of eight.
- `fld_stride` Stride, in bytes, between adjacent rows in a field in both the current block and reference block. `fld_stride` must be a multiple of eight.

**Return Values** The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes** See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoAddBlock_U8_S16(3MLIB)`, `mlib_VideoCopyRef_S16_U8(3MLIB)`,
- `mlib_VideoCopyRef_S16_U8_16x16(3MLIB)`, `mlib_VideoCopyRef_U8_U8(3MLIB)`,
- `mlib_VideoCopyRef_U8_U8_16x16(3MLIB)`, `mlib_VideoCopyRefAve_U8_U8(3MLIB)`,
- `mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB)`,
- `mlib_VideoH263OverlappedMC_U8_U8(3MLIB)`,
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Name  mlib_VideoInterpX_Y_XY_U8_U8 — half-pixel interpolation in both X and Y directions for replenishment mode

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
           #include <mlib.h>

           mlib_status mlib_VideoInterpX_Y_XY_U8_U8(mlib_u8 *outputX,
               mlib_u8 *outputY, mlib_u8 *outputXY, const mlib_u8 *image,
               mlib_s32 stride, mlib_s32 width, mlib_s32 height);

Description  The mlib_VideoInterpX_Y_XY_U8_U8() function performs half-pixel interpolation in both X and Y directions for the replenishment mode.

Parameters  The function takes the following arguments:

outputX  Pointer to the output of X-interpolation. outputX must be 8-byte aligned.
outputY  Pointer to the output of Y-interpolation. outputY must be 8-byte aligned.
outputXY  Pointer to the output of XY-interpolation. outputXY must be 8-byte aligned.
image  Pointer to the image data. image must be 8-byte aligned
stride  Stride, in bytes, between adjacent rows in the image. stride must be a multiple of eight.
width  Width of the image. width must be a multiple of eight.
height  Height of the image. height must be a multiple of two.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoInterpX_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), attributes(5)
### Synopsis

```c
#include <mlib.h>

mlib_status mlib_VideoInterpY_S16_U8_16x16(mlib_s16 *mc_block, const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
mlib_status mlib_VideoInterpY_S16_U8_16x8(mlib_s16 *mc_block, const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
mlib_status mlib_VideoInterpY_S16_U8_8x16(mlib_s16 *mc_block, const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
mlib_status mlib_VideoInterpY_S16_U8_8x8(mlib_s16 *mc_block, const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
mlib_status mlib_VideoInterpY_S16_U8_8x4(mlib_s16 *mc_block, const mlib_u8 *ref_block, mlib_s32 frm_stride, mlib_s32 fld_stride);
```

### Description
Each of these functions performs half-pixel interpolation in the Y direction for a reference block of data type mlib_u8 and a current block of data type mlib_s16. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

### Parameters
Each of the functions takes the following arguments:

- **mc_block**: Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.
- **ref_block**: Pointer to the reference block.
- **frm_stride**: Stride, in bytes, between adjacent rows in a frame in the reference block. frm_stride must be a multiple of eight.
- **fld_stride**: Stride, in bytes, between adjacent rows in a field in the reference block. fld_stride must be a multiple of eight.

### Return Values
Each of the functions returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

### Attributes
See attributes(5) for descriptions of the following attributes:
See Also

mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
mlib_VideoInterpY_S16_U8

**Name**  
mlib_VideoInterpY_S16_U8 – half-pixel interpolation in the Y direction

**Synopsis**  
cc [ flag ... ] file ... -mlib [ library ... ]  
#include <mlib.h>  
mlib_status mlib_VideoInterpY_S16_U8(mlib_s16 *mc_block,  
        const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,  
        mlib_s32 frm_stride, mlib_s32 fld_stride);

**Description**  
The mlib_VideoInterpY_S16_U8() function performs half-pixel interpolation in the Y direction for a reference block of data type mlib_u8 and a current block of data type mlib_s16. In this mode, the output of this function must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block.

**Parameters**  
The function takes the following arguments:

- **mc_block**  
  Pointer to the motion-compensated reference block. mc_block must be 8-byte aligned.

- **ref_block**  
  Pointer to the reference block.

- **width**  
  Width of the blocks.

- **height**  
  Height of the blocks.

- **frm_stride**  
  Stride, in bytes, between adjacent rows in a frame in the reference block. frm_stride must be a multiple of eight.

- **fld_stride**  
  Stride, in bytes, between adjacent rows in a field in the reference block. fld_stride must be a multiple of eight.

**Return Values**  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),  
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),  
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),  
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),  
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),  
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpX_S16_U8(3MLIB)

mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),
mlib_VideoInterpY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Each of these functions performs half-pixel interpolation in the Y direction for a reference block of data type `mlib_u8` and a current block of data type `mlib_u8`. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

**Parameters**

- `curr_block` Pointer to the current block. `curr_block` must be 8-byte aligned.
- `ref_block` Pointer to the reference block.
- `frm_stride` Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. `frm_stride` must be a multiple of eight.
- `fld_stride` Stride, in bytes, between adjacent rows in a field in both the current block and reference block. `fld_stride` must be a multiple of eight.

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
</tbody>
</table>
### mlib_VideoInterpY_U8_U8_16x16(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),  
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),  
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8(3MLIB),  
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),  
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),  
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),  
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),  
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpXY_S16_U8(3MLIB),  
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),  
mlib_VideoInterpXY_U8_U8_16x16(3MLIB), mlib_VideoInterpY_S16_U8(3MLIB),  
mlib_VideoInterpY_S16_U8_16x16(3MLIB), mlib_VideoInterpY_U8_U8(3MLIB),  
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Decimate_U8_U8(3MLIB), attributes(5)
mlib_VideoInterpY_U8_U8(3MLIB)

Name  mlib_VideoInterpY_U8_U8 – half-pixel interpolation in the Y direction

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoInterpY_U8_U8(mlib_u8 *curr_block,
          const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
          mlib_s32 frm_stride, mlib_s32 fld_stride);

Description  The mlib_VideoInterpY_U8_U8() function performs half-pixel interpolation in the Y direction for a reference block of data type mlib_u8 and a current block of data type mlib_u8. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Parameters  The function takes the following arguments:

- **curr_block**  Pointer to the current block. curr_block must be 8-byte aligned.
- **ref_block**  Pointer to the reference block.
- **width**  Width of the blocks.
- **height**  Height of the blocks.
- **frm_stride**  Stride, in bytes, between adjacent rows in a frame in both the current block and reference block. frm_stride must be a multiple of eight.
- **fld_stride**  Stride, in bytes, between adjacent rows in a field in both the current block and reference block. fld_stride must be a multiple of eight.

Return Values  The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8(3MLIB),
mlib_VideoCopyRef_U8_U8_16x16(3MLIB), mlib_VideoCopyRefAve_U8_U8_U8(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB), mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8_U8(3MLIB),
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8_U8_U8(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8_U8(3MLIB),
mlib_VideoInterpX_U8_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8_U8(3MLIB),
mlib_VideoInterpX_U8_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8_U8(3MLIB),
mlib_VideoInterpX_U8_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8_U8(3MLIB),
Name  
mlib_VideoP64Decimate_U8_U8 – averages the source raster image over 2x2 blocks and writes the results to the destination raster image

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoP64Decimate_U8_U8(mlib_u8 *dst,
    const mlib_u8 *src, mlib_s32 width, mlib_s32 height,
    mlib_s32 dst_stride, mlib_s32 src_stride);

Description  
The mlib_VideoP64Decimate_U8_U8() function averages the source raster image over 2x2 blocks and writes the results to the destination raster image. This function is used when the remote side is only capable of QCIF and our scanned image is source to the encoder in CIF format.

Parameters  
The function takes the following arguments:

    dst  Pointer to the destination raster image.
    src  Pointer to the source raster image.
    width  Width of the image.
    height  Height of the image.
    dst_stride  Stride, in bytes, between adjacent rows in the destination image.
    src_stride  Stride, in bytes, between adjacent rows in the source image.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_S16_U8_16x16(3MLIB), mlib_VideoInterpXY_U8_U8(3MLIB),
mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), mlib_VideoP64loop_S16_U8(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB), attributes(5)
Name  
mlib_VideoP64Loop_S16_U8 – applies a 2-dimensional (2D) 3x3 spatial filter on the reference block

Synopsis  
cc [ flag... ] file... -lmilib [ library... ]  
#include <mlib.h>  
mlib_status mlib_VideoP64Loop_S16_U8(mlib_s16 mc_block[64],  
const mlib_u8 *ref_block, mlib_s32 stride);

Description  
The mlib_VideoP64Loop_S16_U8() function applies a 2-dimensional (2D) 3x3 spatial filter on the reference block. The filter is separable into 1D horizontal and vertical functions, where the filter coefficients are 0.25, 0.5, 0.25, except at the block edges where the coefficients are 0, 1, 0. In this mode, the output must be added to the IDCT output to reconstruct the block in the current frame. Thus, the stride applies only to the input reference block. This function requires the motion-compensated block to be 8-bit aligned.

Parameters  
The function takes the following arguments:
- mc_block  Pointer to the motion-compensated reference block.
- ref_block  Pointer to the reference block.
- stride  Stride, in bytes, between adjacent rows in the reference block.

Return Values  
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),  
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),  
mlib_VideoCopyRefAve_U8_U8_16x16(3MLIB),  
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),  
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveX_U8_U8(3MLIB),  
mlib_VideoInterpAveX_U8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8(3MLIB),  
mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB), mlib_VideoInterpAveY_U8_U8(3MLIB),  
mlib_VideoInterpAveY_U8_U8_16x16(3MLIB), mlib_VideoInterpX_S16_U8_16x16(3MLIB),  
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),  
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),  
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),  
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),  
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoP64Loop_S16_U8(3MLIB)

mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_U8_U8(3MLIB),
attributes(5)
Name  mlib_VideoP64Loop_U8_U8 – applies a 2-dimensional (2D) 3x3 spatial filter on the reference block

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoP64Loop_U8_U8(mlib_u8 *curr_block,
                        const mlib_u8 *ref_block, mlib_s32 stride);

Description  The mlib_VideoP64Loop_U8_U8() function applies a 2-dimensional (2D) 3x3 spatial filter on the reference block. The filter is separable into 1D horizontal and vertical functions, where the filter coefficients are 0.25, 0.5, 0.25, except at the block edges where the coefficients are 0, 1, 0. In this mode, the motion-compensated reference block becomes the current block. Thus, the stride applies to both the input reference block and the current block.

Parameters  The function takes the following arguments:

curr_block  Pointer to the current block.

ref_block  Pointer to the reference block.

stride  Stride, in bytes, between adjacent rows in both the current block and the reference block.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoAddBlock_U8_S16(3MLIB), mlib_VideoCopyRef_S16_U8(3MLIB),
mlib_VideoCopyRef_S16_U8_16x16(3MLIB), mlib_VideoCopyRef_U8_U8_16x16(3MLIB),
mlib_VideoH263OverlappedMC_S16_U8(3MLIB),
mlib_VideoH263OverlappedMC_U8_U8(3MLIB), mlib_VideoInterpAveU8_U8(3MLIB),
mlib_VideoInterpAveU8_U8_16x16(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveXY_U8_U8(3MLIB), mlib_VideoInterpAveXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpAveY_U8_U8(3MLIB), mlib_VideoInterpAveY_U8_U8_16x16(3MLIB),
mlib_VideoInterpX_S16_U8_16x16(3MLIB), mlib_VideoInterpX_U8_U8(3MLIB),
mlib_VideoInterpXY_S16_U8(3MLIB), mlib_VideoInterpXY_S16_U8_16x16(3MLIB),
mlib_VideoInterpXY_U8_U8(3MLIB), mlib_VideoInterpXY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_S16_U8(3MLIB), mlib_VideoInterpY_S16_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoInterpY_U8_16x16(3MLIB), mlib_VideoInterpY_U8_16x16(3MLIB),
mlib_VideoP64Loop_U8_U8(3MLIB)

mlib_VideoInterpY_U8_U8(3MLIB), mlib_VideoInterpY_U8_U8_16x16(3MLIB),
mlib_VideoP64Decimate_U8_U8(3MLIB), mlib_VideoP64Loop_S16_U8(3MLIB),
attributes(5)
Name  mlib_VideoQuantizeInit_S16 – quantization of forward Discrete Cosine Transform (DCT) coefficients

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoQuantizeInit_S16(mlib_d64 dqtable[64],
                                             const mlib_s16 iqtable[64]);

Description  The mlib_VideoQuantizeInit_S16() function initializes the quantization table.

The following equation is used:

dqtable[i] = 1.0 / iqtable[i]; 0 ≤ i < 64

Parameters  The function takes the following arguments:

dqtable  Pointer to quantizer table coefficients.

iqtable  Pointer to original quantizer table coefficients:

0 < iqtable[i] < 128

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB),
mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB),
mlib_VideoDCT8x8_S16_S16_NA(3MLIB), mlib_VideoDCT8x8_S16_U8(3MLIB),
mlib_VideoDCT8x8_S16_U8_NA(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB),
mlib_VideoDCT16x16_S16_S16_B10(3MLIB), mlib_VideoDCT16x16_S16_U8(3MLIB),
mlib_VideoDCT16x16_S16_U8_NA(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB),
mlib_VideoDeQuantize_S16(3MLIB), attributes(5)
The `mlib_VideoQuantize_S16()` function performs quantization on DCT coefficients. The following equation is used:

\[ \text{icoeffs}[i] = \text{icoeffs}[i] \times \text{dqtable}[i]; \quad 0 \leq i < 64 \]

**Parameters**
The function takes the following arguments:

- `icoeffs`  
  Pointer to the output DCT coefficients:
  
  \[-2048 < \text{icoeffs}[i] < 2048\]
  
  Note that `icoeffs` must be 8-byte aligned.

- `dqtable`  
  Pointer to quantizer table coefficients.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoDCT2x2_S16_S16(3MLIB), mlib_VideoDCT4x4_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_S16(3MLIB), mlib_VideoDCT8x8_S16_B12(3MLIB), mlib_VideoDCT8x8_S16_S16_B12(3MLIB), mlib_VideoDCT16x16_S16_S16(3MLIB), mlib_VideoDCT16x16_S16_B10(3MLIB), mlib_VideoDCT16x16_S16_B12(3MLIB), mlib_VideoDeQuantize_S16(3MLIB), mlib_VideoDeQuantizeInit_S16(3MLIB), mlib_VideoQuantizeInit_S16(3MLIB), attributes(5)
mlib_VideoReversibleColorRGB2YUV_U8_U8,
mlib_VideoReversibleColorRGB2YUV_S16_U8,
mlib_VideoReversibleColorRGB2YUV_S16_S16,
mlib_VideoReversibleColorRGB2YUV_S32_S16 – reversible color space conversion for
wavelet transformation

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoReversibleColorRGB2YUV_U8_U8(mlib_u8 *y,
    mlib_u8 *u, mlib_u8 *v, const mlib_u8 *r, const mlib_u8 *g,
    const mlib_u8 *b, mlib_s32 n, mlib_s32 depth);

mlib_status mlib_VideoReversibleColorRGB2YUV_S16_U8(mlib_s16 *y,
    mlib_s16 *u, mlib_s16 *v, const mlib_u8 *r, const mlib_u8 *g,
    const mlib_u8 *b,
    mlib_s32 n, mlib_s32 depth);

mlib_status mlib_VideoReversibleColorRGB2YUV_S16_S16(mlib_s16 *y,
    mlib_s16 *u, mlib_s16 *v, const mlib_s16 *r, const mlib_s16 *g,
    const mlib_s16 *b, mlib_s32 n, mlib_s32 depth);

mlib_status mlib_VideoReversibleColorRGB2YUV_S32_S16(mlib_s32 *y,
    mlib_s32 *u, mlib_s32 *v, const mlib_s16 *r, const mlib_s16 *g,
    const mlib_s16 *b, mlib_s32 n, mlib_s32 depth);

Each of the functions provides support to reversible wavelet transformation. It is for reversible
color space conversion.

Each of the functions takes the following arguments:

- **y** Pointer to destination Y component.
- **u** Pointer to destination U component.
- **v** Pointer to destination V component.
- **r** Pointer to source R component.
- **g** Pointer to source G component.
- **b** Pointer to source B component.
- **n** Length of data.
- **depth** Number of bit planes required to store the original R, G, and B components.

Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

See attributes(5) for descriptions of the following attributes:
### mlib_VideoReversibleColorRGB2YUV_U8_U8(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoReversibleColorYUV2RGB_U8_U8(3MLIB), attributes(5)
mlib_VideoReversibleColorYUV2RGB_U8_U8
mlib_VideoReversibleColorYUV2RGB_U8_S16
mlib_VideoReversibleColorYUV2RGB_S16_S16
mlib_VideoReversibleColorYUV2RGB_S16_S32 – reversible color space conversion for wavelet transformation

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoReversibleColorYUV2RGB_U8_U8(mlib_u8 *r,
mlib_u8 *g, mlib_u8 *b, const mlib_u8 *y, const mlib_u8 *u,
const mlib_u8 *v, mlib_s32 n, mlib_s32 depth);
mlib_status mlib_VideoReversibleColorYUV2RGB_U8_S16(mlib_u8 *r,
mlib_u8 *g, mlib_u8 *b, const mlib_s16 *y, const mlib_s16 *u,
const mlib_s16 *v, mlib_s32 n, mlib_s32 depth);
mlib_status mlib_VideoReversibleColorYUV2RGB_S16_S16(mlib_s16 *r,
mlib_s16 *g, mlib_s16 *b, const mlib_s16 *y, const mlib_s16 *u,
const mlib_s16 *v, mlib_s32 n, mlib_s32 depth);
mlib_status mlib_VideoReversibleColorYUV2RGB_S16_S32(mlib_s16 *r,
mlib_s16 *g, mlib_s16 *b, const mlib_s32 *y, const mlib_s32 *u,
const mlib_s32 *v, mlib_s32 n, mlib_s32 depth);

Description Each of the functions provides support to reversible wavelet transformation. It is for reversible color space conversion.

Parameters Each of the functions takes the following arguments:

- r Pointer to destination R component.
- g Pointer to destination G component.
- b Pointer to destination B component.
- y Pointer to source Y component.
- u Pointer to source U component.
- v Pointer to source V component.
- n Length of data.
- depth Number of bit planes required to store the original R, G, and B components.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:
mlib_VideoReversibleColorYUV2RGB_U8_U8(3MLIB)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoReversibleColorRGB2YUV_U8_U8(3MLIB), attributes(5)
mlib_VideoSignMagnitudeConvert_S16 – wavelet transformation, sign-magnitude conversion

Synopsis

```
#include <mlib.h>

mlib_status mlib_VideoSignMagnitudeConvert_S16(mlib_s16 *srcdst, mlib_s32 n);
```

Description

The `mlib_VideoSignMagnitudeConvert_S16()` function converts data between standard 2's complement signed integer representation and sign-magnitude representation.

Parameters

The function takes the following arguments:

- `dst` Pointer to destination data array.
- `src` Pointer to source data array.
- `srcdst` Pointer to source and destination data array.
- `n` Array size.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoSignMagnitudeConvert_S16_S16(3MLIB)`
- `mlib_VideoSignMagnitudeConvert_S32(3MLIB)`
- `mlib_VideoSignMagnitudeConvert_S32_S32(3MLIB), attributes(5)`
mlib_VideoSignMagnitudeConvert_S16_S16 – wavelet transformation, sign-magnitude conversion

Synopsis

```c
#include <mlib.h>

mlib_status mlib_VideoSignMagnitudeConvert_S16_S16(mlib_s16 *dst,
    const mlib_s16 *src, mlib_s32 n);
```

Description

The `mlib_VideoSignMagnitudeConvert_S16_S16()` function converts data between standard 2s complement signed integer representation and sign-magnitude representation.

Parameters

- `dst` Pointer to destination data array.
- `src` Pointer to source data array.
- `srcdst` Pointer to source and destination data array.
- `n` Array size.

Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

- `mlib_VideoSignMagnitudeConvert_S16(3MLIB)`
- `mlib_VideoSignMagnitudeConvert_S32(3MLIB)`
- `attributes(5)`
The `mlib_VideoSignMagnitudeConvert_S32()` function converts data between standard 2's complement signed integer representation and sign-magnitude representation.

**Parameters**

The function takes the following arguments:

- `dst` Pointer to destination data array.
- `src` Pointer to source data array.
- `srcdst` Pointer to source and destination data array.
- `n` Array size.

**Return Values**

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

`mlib_VideoSignMagnitudeConvert_S16(3MLIB)`, `mlib_VideoSignMagnitudeConvert_S16_S16(3MLIB)`, `mlib_VideoSignMagnitudeConvert_S32_S32(3MLIB)`, `attributes(5)`
mlib_VideoSignMagnitudeConvert_S32_S32 – wavelet transformation, sign-magnitude conversion

Synopsis
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoSignMagnitudeConvert_S32_S32(mlib_s32 *dst,
const mlib_s32 *src, mlib_s32 n);

Description
The mlib_VideoSignMagnitudeConvert_S32_S32() function converts data between standard 2s complement signed integer representation and sign-magnitude representation.

Parameters
The function takes the following arguments:

dst Pointer to destination data array.
src Pointer to source data array.
srccdst Pointer to source and destination data array.
n Array size.

Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also
mlib_VideoSignMagnitudeConvert_S16(3MLIB),
mlib_VideoSignMagnitudeConvert_S16_S16(3MLIB),
mlib_VideoSignMagnitudeConvert_S32(3MLIB), attributes(5)
mlib_VideoSumAbsDiff - motion estimation

Synopsis

```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_s32 mlib_VideoSumAbsDiff(mlib_u8 *curr_block,
    const mlib_u8 *ref_block, mlib_s32 width, mlib_s32 height,
    mlib_s32 stride);
```

Description

The `mlib_VideoSumAbsDiff()` function computes the sum of absolute differences between the pixels in the current block and the corresponding pixels in the reference block.

Both the current block and the reference block belong to frames with the same dimension. (The stride is applicable to both.) Motion estimation computes the sum of the absolute differences between the current block and reference blocks at different locations in the reference frame, choosing the best fit (least sum of absolute difference) to calculate the motion vector.

Parameters

- `curr_block`: Pointer to the current block. `curr_block` must be 8-byte aligned.
- `ref_block`: Pointer to the reference block.
- `width`: Width of the block.
- `height`: Height of the block.
- `stride`: Stride, in bytes, between adjacent rows in a block. `stride` must be a multiple of eight.

Return Values

The function returns a value of type `mlib_s32`.

Attributes

See `attributes(5)` for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also

`attributes(5)`
The `mlib_VideoUpSample420()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters**
- `dst0` Pointer to upper destination row. `dst0` must be 8-byte aligned.
- `dst1` Pointer to lower destination row. `dst1` must be 8-byte aligned.
- `src0` Pointer to upper source row. `src0` must be 8-byte aligned.
- `src1` Pointer to middle source row. `src1` must be 8-byte aligned.
- `src2` Pointer to lower source row. `src2` must be 8-byte aligned.
- `n` Length of source rows. `n` must be greater than 1.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- `mlib_VideoDownSample420(3MLIB)`
- `mlib_VideoDownSample420_S16(3MLIB)`
- `mlib_VideoDownSample422(3MLIB)`
- `mlib_VideoDownSample422_S16(3MLIB)`
- `mlib_VideoUpSample420_Nearest(3MLIB)`
- `mlib_VideoUpSample420_Nearest_S16(3MLIB)`
- `mlib_VideoUpSample422(3MLIB)`
- `mlib_VideoUpSample422_S16(3MLIB)`
- `mlib_VideoUpSample422_Nearest(3MLIB)`
- `mlib_VideoUpSample422_Nearest_S16(3MLIB)`
- attributes(5)
mlib_VideoUpSample420_Nearest(3MLIB)

Name  mlib_VideoUpSample420_Nearest – up sampling rate conversion in JFIF

Synopsis  cc [ flag... ] file... -mlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoUpSample420_Nearest(mlib_u8 *dst0, mlib_u8 *dst1,
const mlib_u8 *src, mlib_s32 n);

Description  The mlib_VideoUpSample420_Nearest() function performs up sampling rate conversion
used in JPEG File Interchange Format (JFIF).

Parameters  The function takes the following arguments:
 dst0 Pointer to upper destination row. dst0 must be 8-byte aligned.
 dst1 Pointer to lower destination row. dst1 must be 8-byte aligned.
 src Pointer to source row. src must be 8-byte aligned.
 n Length of source rows. n must be greater than 1.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoDownSample420(3MLIB), mlib_VideoDownSample420_S16(3MLIB),
mlib_VideoDownSample422(3MLIB), mlib_VideoDownSample422_S16(3MLIB),
mlib_VideoUpSample420(3MLIB), mlib_VideoUpSample420_Nearest_S16(3MLIB),
mlib_VideoUpSample420_S16(3MLIB), mlib_VideoUpSample422(3MLIB),
mlib_VideoUpSample422_S16(3MLIB), mlib_VideoUpSample422_Nearest(3MLIB),
mlib_VideoUpSample422_Nearest_S16(3MLIB), attributes(5)

Multimedia Library Functions - Part 7  1505
The `mlib_VideoUpSample420_Nearest_S16()` function performs upsampling rate conversion used in JPEG File Interchange Format (JFIF).

### Parameters

- **dst0**: Pointer to upper destination row. `dst0` must be 8-byte aligned.
- **dst1**: Pointer to lower destination row. `dst1` must be 8-byte aligned.
- **src**: Pointer to source row. `src` must be 8-byte aligned.
- **n**: Length of source rows. `n` must be greater than 1.

### Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise, it returns `MLIB_FAILURE`.

### Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

- `mlib_VideoDownSample420(3MLIB)`, `mlib_VideoDownSample420_S16(3MLIB)`, `mlib_VideoDownSample422(3MLIB)`, `mlib_VideoDownSample422_S16(3MLIB)`, `mlib_VideoUpSample420(3MLIB)`, `mlib_VideoUpSample420_Nearest(3MLIB)`, `mlib_VideoUpSample420_S16(3MLIB)`, `mlib_VideoUpSample422(3MLIB)`, `mlib_VideoUpSample422_Nearest(3MLIB)`, `mlib_VideoUpSample422_S16(3MLIB)`, `attributes(5)`
# mlib_VideoUpSample420_S16

## Synopsis

```c
#include <mlib.h>

mlib_status mlib_VideoUpSample420_S16(mlib_s16 *dst0,
                                       mlib_s16 *dst1,
                                       const mlib_s16 *src0,
                                       const mlib_s16 *src1,
                                       const mlib_s16 *src2, mlib_s32 n);
```

## Description

The `mlib_VideoUpSample420_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

## Parameters

- `dst0`: Pointer to upper destination row. `dst0` must be 8-byte aligned.
- `dst1`: Pointer to lower destination row. `dst1` must be 8-byte aligned.
- `src0`: Pointer to upper source row. `src0` must be 8-byte aligned.
- `src1`: Pointer to middle source row. `src1` must be 8-byte aligned.
- `src2`: Pointer to lower source row. `src2` must be 8-byte aligned.
- `n`: Length of source rows. `n` must be greater than 1.

## Return Values

The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

## Attributes

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

## See Also

- `mlib_VideoDownSample420(3MLIB)`, `mlib_VideoDownSample420_S16(3MLIB)`,
- `mlib_VideoDownSample422(3MLIB)`, `mlib_VideoDownSample422_S16(3MLIB)`,
- `mlib_VideoUpSample420(3MLIB)`, `mlib_VideoUpSample420_Nearest(3MLIB)`,
- `mlib_VideoUpSample420_Nearest_S16(3MLIB)`, `mlib_VideoUpSample422(3MLIB)`,
- `mlib_VideoUpSample422_Nearest(3MLIB)`, `mlib_VideoUpSample422_Nearest_S16(3MLIB)`,
- attributes(5)
The `mlib_VideoUpSample422()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters**
The function takes the following arguments:

- `dst` Pointer to destination row. `dst` must be 8-byte aligned.
- `src` Pointer to source row. `src` must be 8-byte aligned.
- `n` Length of source rows. `n` must be greater than 1.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See **attributes(5)** for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
`mlib_VideoDownSample420(3MLIB), mlib_VideoDownSample420_S16(3MLIB), mlib_VideoDownSample422(3MLIB), mlib_VideoDownSample422_S16(3MLIB), mlib_VideoUpSample420(3MLIB), mlib_VideoUpSample420_Nearest(3MLIB), mlib_VideoUpSample420_Nearest_S16(3MLIB), mlib_VideoUpSample422(3MLIB), mlib_VideoUpSample422_S16(3MLIB), mlib_VideoUpSample422_Nearest(3MLIB), mlib_VideoUpSample422_Nearest_S16(3MLIB), attributes(5)`
mlib_VideoUpSample422_Nearest

Name  mlib_VideoUpSample422_Nearest – up sampling rate conversion in JFIF

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoUpSample422_Nearest(mlib_u8 *dst, const mlib_u8 *src,
mlib_s32 n);

Description  The mlib_VideoUpSample422_Nearest() function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

Parameters  The function takes the following arguments:

- **dst**  Pointer to destination row. dst must be 8-byte aligned.
- **src**  Pointer to source row. src must be 8-byte aligned.
- **n**  Length of source rows. n must be greater than 1.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VideoDownSample420(3MLIB), mlib_VideoDownSample420_S16(3MLIB),
mlib_VideoDownSample422(3MLIB), mlib_VideoDownSample422_S16(3MLIB),
mlib_VideoUpSample420(3MLIB), mlib_VideoUpSample420_Nearest(3MLIB),
mlib_VideoUpSample420_Nearest_S16(3MLIB), mlib_VideoUpSample420_S16(3MLIB),
mlib_VideoUpSample422(3MLIB), mlib_VideoUpSample422_S16(3MLIB),
mlib_VideoUpSample422_Nearest(3MLIB), mlib_VideoUpSample422_Nearest_S16(3MLIB), attributes(5)
**mlib_VideoUpSample422_Nearest_S16**

**Name**
mlib_VideoUpSample422_Nearest_S16 – up sampling rate conversion in JFIF

**Synopsis**
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoUpSample422_Nearest_S16(mlib_s16 *dst,
const mlib_s16 *src, mlib_s32 n);
```

**Description**
The `mlib_VideoUpSample422_Nearest_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters**
The function takes the following arguments:
- `dst` Pointer to destination row. `dst` must be 8-byte aligned.
- `src` Pointer to source row. `src` must be 8-byte aligned.
- `n` Length of source rows. `n` must be greater than 1.

**Return Values**
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**
- mlib_VideoDownSample420(3MLIB), mlib_VideoDownSample420_S16(3MLIB),
- mlib_VideoDownSample422(3MLIB), mlib_VideoDownSample422_S16(3MLIB),
- mlib_VideoUpSample420(3MLIB), mlib_VideoUpSample420_Nearest(3MLIB),
- mlib_VideoUpSample420_S16(3MLIB), mlib_VideoUpSample420_S16(3MLIB),
- mlib_VideoUpSample422(3MLIB), mlib_VideoUpSample422_S16(3MLIB),
- mlib_VideoUpSample422_Nearest(3MLIB), attributes(5)
mlib_VideoUpSample422_S16(3MLIB)

**Name**  
mlib_VideoUpSample422_S16 – up sampling rate conversion in JFIF

**Synopsis**  
cc [ flag... ] file... -lmlib [ library... ]  
#include <mlib.h>

mlib_status mlib_VideoUpSample422_S16(mlib_s16 *dst,  
    const mlib_s16 *src, mlib_s32 n);

**Description**  
The `mlib_VideoUpSample422_S16()` function performs up sampling rate conversion used in JPEG File Interchange Format (JFIF).

**Parameters**  
The function takes the following arguments:

- **dst**  
  Pointer to destination row. `dst` must be 8-byte aligned.

- **src**  
  Pointer to source row. `src` must be 8-byte aligned.

- **n**  
  Length of source rows. `n` must be greater than 1.

**Return Values**  
The function returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**  
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
mlib_VideoDownSample420(3MLIB), mlib_VideoDownSample420_S16(3MLIB),  
mlib_VideoDownSample422(3MLIB), mlib_VideoDownSample422_S16(3MLIB),  
mlib_VideoUpSample420(3MLIB), mlib_VideoUpSample420_Nearest(3MLIB),  
mlib_VideoUpSample420_Nearest_S16(3MLIB), mlib_VideoUpSample420_S16(3MLIB),  
mlib_VideoUpSample422(3MLIB), mlib_VideoUpSample422_S16(3MLIB),  
mlib_VideoUpSample422_Nearest(3MLIB), mlib_VideoUpSample422_Nearest_S16(3MLIB)
mlib_VideoWaveletForwardTwoTenTrans(3MLIB)

Name mlib_VideoWaveletForwardTwoTenTrans,
     mlib_VideoWaveletForwardTwoTenTrans_S16_U8,
     mlib_VideoWaveletForwardTwoTenTrans_S16_S16,
     mlib_VideoWaveletForwardTwoTenTrans_S32_S16,
     mlib_VideoWaveletForwardTwoTenTrans_S32_S32 – wavelet transformation

Synopsis cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoWaveletForwardTwoTenTrans_S16_U8(mlib_s16 *dst,
            const mlib_u8 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);
mlib_status mlib_VideoWaveletForwardTwoTenTrans_S16_S16(mlib_s16 *dst,
            const mlib_s16 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);
mlib_status mlib_VideoWaveletForwardTwoTenTrans_S32_S16(mlib_s32 *dst,
            const mlib_s16 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);
mlib_status mlib_VideoWaveletForwardTwoTenTrans_S32_S32(mlib_s32 *dst,
            const mlib_s32 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

Description Each of the functions provides support to reversible wavelet transformation. It is for a forward
two-ten transformation.

Parameters Each of the functions takes the following arguments:

    dst Pointer to TT-transform coefficients.
    src Pointer to source image.
    width Width of image.
    height Height of image.
    level Pointer to the number of decomposition levels. It returns the processed
decomposition levels value.

Return Values Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also mlib_VideoWaveletInverseTwoTenTrans(3MLIB), attributes(5)
Name: mlib_VideoWaveletInverseTwoTenTrans, mlib_VideoWaveletInverseTwoTenTrans_U8_S16, mlib_VideoWaveletInverseTwoTenTrans_S16_S16, mlib_VideoWaveletInverseTwoTenTrans_S16_S32, mlib_VideoWaveletInverseTwoTenTrans_S32_S32 – wavelet transformation

Synopsis:

cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VideoWaveletInverseTwoTenTrans_U8_S16(mlib_u8 *dst, const mlib_s16 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

mlib_status mlib_VideoWaveletInverseTwoTenTrans_S16_S16(mlib_s16 *dst, const mlib_s16 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

mlib_status mlib_VideoWaveletInverseTwoTenTrans_S16_S32(mlib_s16 *dst, const mlib_s32 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

mlib_status mlib_VideoWaveletInverseTwoTenTrans_S32_S32(mlib_s32 *dst, const mlib_s32 *src, mlib_s32 width, mlib_s32 height, mlib_s32 *level);

Description: Each of the functions provides support to reversible wavelet transformation. It is for an inverse two-ten transformation.

Parameters: Each of the functions takes the following arguments:

dst  Pointer to destination image.
src  Pointer to TT-transform coefficients.
width  Width of image.
height  Height of image.
level  Pointer to the number of decomposition levels. It returns the processed decomposition levels value.

Return Values: Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes: See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also: mlib_VideoWaveletForwardTwoTenTrans(3MLIB), attributes(5)
Name  mlib_VolumeFindMaxBMask_U8, mlib_VolumeFindMaxBMask_S16 – maximum intensity searching

Synopsis  cc [ flag... ] file... -mlib [ library... ]
# include <mlib.h>

mlib_status mlib_VolumeFindMaxBMask_U8(mlib_u8 *max,
    const mlib_rays *rays, const mlib_u8 *bmask);
mlib_status mlib_VolumeFindMaxBMask_S16(mlib_s16 *max,
    const mlib_rays *rays, const mlib_u8 *bmask);

Description  Each function performs maximum intensity searching.

It uses the following equation:

max[i] = MAX( rays->results[j][i]
    j = 0, 1, ..., rays->nsteps[i]; bmask[j] = 1 )

where i = 0, 1, ..., rays->nrays - 1.

Parameters  The function takes the following arguments:

max       Pointer to an array of rays->nrays maximum values of the samples in each ray.
rays      Pointer to an mlib_rays structure. The data rays->results are organized with ray number (rather than ray step) varying fastest. Ray number and ray step are the output of the ray casting functions. The data might have values beyond the maximum step on a ray. For example, rays->results[rays->nsteps[i]][i] on ray i might not equal 0.

bmask     Pointer to a 1-bit mask array. Eight mask bits are packed into one byte. A 1 corresponds to the data in the step to be considered.

Return Values  The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

Attributes  See attributes(5) for descriptions of the following attributes:

+-----------------+------------------+
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VolumeFindMax_U8(3MLIB), mlib_VolumeFindMaxCMask_U8(3MLIB), attributes(5)
mlib_VolumeFindMaxCMask_U8, mlib_VolumeFindMaxCMask_S16 – maximum intensity searching

Synopsis  
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VolumeFindMaxCMask_U8(mlib_u8 *max,
   const mlib_rays *rays, const mlib_u8 *cmask, mlib_s32 thresh);
mlib_status mlib_VolumeFindMaxCMask_S16(mlib_s16 *max,
   const mlib_rays *rays, const mlib_u8 *cmask, mlib_s32 thresh);

Description  
Each function performs maximum intensity searching.

It uses the following equation:

\[
max[i] = \max\{ \text{rays->results}[j][i] \mid j = 0, 1, \ldots, \text{rays->nsteps}[i]; \text{cmask}[j] > \text{thresh} \}
\]

where \( i = 0, 1, \ldots, \text{rays->nrays} - 1 \).

Parameters  
The function takes the following arguments:

- \( max \)  
  Pointer to an array of \( \text{rays->nrays} \) maximum values of the samples in each ray.

- \( rays \)  
  Pointer to an mlib_rays structure. The data \( \text{rays->results} \) are organized with ray number (rather than ray step) varying fastest. Ray number and ray step are the output of the ray casting functions. The data might have values beyond the maximum step on a ray. For example, \( \text{rays->results}[\text{rays->nsteps}[i]][i] \) on ray \( i \) might not equal \( 0 \).

- \( cmask \)  
  Pointer to an unsigned 8-bit mask array. If \( \text{cmask}[j] > \text{thresh} \), then data in step \( j \), \( \text{rays->results}[j] \), are considered.

- \( thresh \)  
  Threshold.

Return Values  
The function returns \texttt{MLIB_SUCCESS} if successful. Otherwise it returns \texttt{MLIB_FAILURE}.

Attributes  
See \texttt{attributes(5)} for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  
\texttt{mlib_VolumeFindMax_U8(3MLIB)}, \texttt{mlib_VolumeFindMaxBMask_U8(3MLIB)}, \texttt{attributes(5)}
### Name
mlib_VolumeFindMax_U8, mlib_VolumeFindMax_S16 – maximum intensity searching

### Synopsis
```c
cc [ flag... ] file... -lmlib [ library... ]
#include <mlib.h>

mlib_status mlib_VolumeFindMax_U8(mlib_u8 *max,
    const mlib_rays *rays);
mlib_status mlib_VolumeFindMax_S16(mlib_s16 *max,
    const mlib_rays *rays);
```

### Description
Each function performs maximum intensity searching. It uses the following equation:

\[
\text{max}[i] = \text{MAX} \{ \text{rays->results}[j][i] \}
\]

where \(i = 0, 1, \ldots, \text{rays->nrays} - 1\).

### Parameters
- **max**: Pointer to an array of \text{rays->nrays} maximum values of the samples in each ray.
- **rays**: Pointer to an mlib_rays structure. The data \text{rays->results} are organized with ray number (rather than ray step) varying fastest. Ray number and ray step are the output of the ray casting functions. The data might have values beyond the maximum step on a ray. For example, \text{rays->results[rays->nsteps[i]][i]} on ray \(i\) might not equal 0.

### Return Values
The function returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

### Attributes
See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also
- mlib_VolumeFindMaxBMask_U8(3MLIB), mlib_VolumeFindMaxCMask_U8(3MLIB), attributes(5)
Each of these functions casts a ray (or rays) through a three-dimensional (3D) data set, then computes and returns the interpolated samples at each step along the way.

In trilinear interpolation, the value at point P is computed from its eight surrounding neighbors based on the equation below.

\[
P = (1-a)*(1-b)*(1-c)*P_0 + \\
a*(1-b)*(1-c)*P_x + (1-a)*b*(1-c)*P_y + (1-a)*(1-b)*c*P_z + \\
a*b*(1-c)*P_{xy} + a*(1-b)*c*P_{xz} + (1-a)*b*c*P_{yz} + \\
a*b*c*P_{xyz}
\]

where a, b, and c are the fractional parts of the coordinates of point P.

The trilinear interpolation is represented by the following figure:
In nearest neighbor operation, the sample value at point P is replaced by the value of the nearest neighbor voxel.

**Parameters**  Each of the functions takes the following arguments:
- `rays`  Casting rays.
- `blk`  Volume data in the blocked format.
- `buffer`  Working buffer.

**Return Values**  Each of the functions returns MLIB_SUCCESS if successful. Otherwise it returns MLIB_FAILURE.

**Attributes**  See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See Also  mlib_VolumeRayCast_General(3MLIB), attributes(5)
Each of these functions casts a ray (or rays) through a three-dimensional (3D) data set, then computes and returns the interpolated samples at each step along the way.

In trilinear interpolation, the value at point \( P \) is computed from its eight surrounding neighbors based on the equation below.

\[
P = (1-a)(1-b)(1-c)P_0 + a(1-b)(1-c)Px + (1-a)b(1-c)Py + (1-a)(1-b)cPz +
\]
where $a$, $b$, and $c$ are the fractional parts of the coordinates of point $P$.

The trilinear interpolation is represented by the following figure:

In nearest neighbor operation, the sample value at point $P$ is replaced by the value of the nearest neighbor voxel.

**Parameters**

Each of the functions takes the following arguments:

- `rays`  
  Casting rays.

- `vol`  
  Volume data that consists of slices.

- `buffer`  
  Working buffer.

**Return Values**

Each of the functions returns `MLIB_SUCCESS` if successful. Otherwise it returns `MLIB_FAILURE`.

**Attributes**

See [attributes(5)] for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
See Also  mlib_VolumeRayCast_Blocked(3MLIB), attributes(5)
Name  mlib_VolumeWindowLevel – window-level operation

Synopsis  cc [ flag... ] file... -lmlib [ library... ]
          #include <mlib.h>
          mlib_status mlib_VolumeWindowLevel(mlib_u8 *dst,
          const mlib_s16 *src, mlib_s32 window, mlib_s32 level,
          mlib_s32 gmax, mlib_s32 gmin, mlib_s32 len);

Description  The mlib_VolumeWindowLevel() function performs a window-level operation by using the following equation:

\[
\begin{align*}
X &= x(i) \quad i = 0, 1, \ldots, n - 1 \\
Z &= z(i) \quad i = 0, 1, \ldots, n - 1 \\
    &= \begin{cases} 
        g_{\text{min}} & x(i) < l - \frac{w}{2} \\
        \frac{x(i) - \left(l - \frac{w}{2}\right)}{w} + g_{\text{min}} & l - \frac{w}{2} \leq x(i) < l + \frac{w}{2} \\
        g_{\text{max}} & x(i) \geq l + \frac{w}{2}
    \end{cases}
\end{align*}
\]

The window-level operation is represented by the following figure:
The function takes the following arguments:

- **dst**: Pointer to the output or destination array.
- **src**: Pointer to the input or source array.
- **window**: Width of the window.
- **level**: Center of the window.
- **gmax**: Maximum grayscale in the destination array.
- **gmin**: Minimum grayscale in the destination array.
- **len**: Length of the data array.

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Parameters**

**Return Values**

The function returns **MLIB_SUCCESS** if successful. Otherwise it returns **MLIB_FAILURE**.

**Attributes**

See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Committed</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also** attributes(5)