

AccuSoft ImageGear v15.3

For all UNIX Platforms
For AIX, HP-UX, Linux, Solaris



Copyright Information

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of AccuSoft® Corporation.

© 1996-2007 AccuSoft® Corporation. All rights reserved.

AccuSoft and ImageGear are registered trademarks of AccuSoft Corporation. High Performance Imaging and the “half gear” logo are trademarks of AccuSoft Corporation. All rights reserved. All other company and brand names are trademarks or registered trademarks of their respective owners.

This manual and the software described in it are both products of the USA.

Document revision number: **1-17-08**

AccuSoft Corporation, 71 Lyman Street, Northborough, MA 01532

(508) 351-9092,

(508) 351-9086 (Fax)

Internet address: **www.accusoft.com**

Table of Contents

Copyright Information 2

Volume 1: User Guide 33

Introduction 35

In This Guide 35

What's New in ImageGear 36
What's New in ImageGear 15.3 37

ImageGear Overview 39
High Speed Display. 40
Image Loading and Saving 41
Printing 42
Image Processing 42
Special Effects 43
Pixel Access 43
ROI Processing. 43
Format Conversion 43
Compression. 44
Transitions. 44
Native CMYK Support. 45
Database. 45
Annotations 45
Built-in Internet Support (URL, FTP, Gopher). 45

ImageGear Supported File Formats 46

ImageGear Support for DCRAW Formats 48

ImageGear Components 49
ImageGear Component Descriptions 49

ImageGear Samples 51

Answering Your ImageGear Questions 51
AccuSoft Customer Support 52
Before Contacting Customer Support 52

Internet Services 52

Installation 53

Minimum Requirements 53
Vista and Visual Studio 2005 53

Installing ImageGear from the Web Site 55
Directory Structure 59
Description of Installed Files 60
Sample Applications 65
Uninstalling ImageGear 65
ImageGear Licensing 66

Getting Started 69

Converting an Image 69
Loading and Displaying an Image 71
Converting an Image to DDB and Displaying It 72
Displaying an Image with Scroll Bars 74

Understanding Bitmap Images 77

Pixels 77
Channels 77
Color Spaces 78
24-bit RGB Images 78
1-bit Images 79
4-bit and 8-bit Images 79
Gray-Scale Images 80
Color Values Used During Display 81
Device-Independent/Device-Dependent Bitmaps 81
Device-Independent Bitmaps (DIB) 81
Device-Dependent Bitmaps (DDBs) 83
Vector Images 83
ImageGear Architecture Diagram 84

Using ImageGear 85

Core Component Function Naming Conventions 86
How to Call an ImageGear Function 87
ImageGear Data Types and Structures 88
Data Types 88
Data Structures 89
UNIX Platform Specific Functions and Features 92

ImageGear Licensing API 92
Loading Images 93
Working With Color Space Options. 94
Saving Images 95
The Image Rectangle 95
Saving Images to a Disk File. 96
Saving to a Disk File Using a File Descriptor Handle 96
Saving an Image to Memory 98
Converting Images from One File Format to Another 100
Displaying Images 101
Concepts 101
Understanding Storage Options 102
Understanding Display Options. 102
Image Processing 114
Color Reduction 115
Color Promotion 116
Contrast Alteration 118
Image Transformation 119
Image Analysis 120
Special Effects 121
Blending and Combining Images 122
Image Correction. 123
Image Maintenance. 123
Pixel Access Operations 124
Allocating Space for ImageGear Pixel Access Functions 125
Getting and Setting Individual Pixels. 125
Getting and Setting Linear Groups of Pixels 126
Getting and Setting a Rectangular Area of Pixels. 127
Using Pixel Access Functions with ImageGear. 127
Clipboard operations 128
Copying/Cutting to the Clipboard 128
Checking the Contents of the Clipboard 128
Pasting an Image from the Clipboard 129
Printing Images 130
Thread Safety 130
Working with Format Filters 131
Using the Format Filters API for Image Loading and Getting Information. . . 132
Using Format Filters API for Image Saving. 134
Using Format Filters API for Filter Control 135
Non-Image Data Processing 137
Working with ImageGear Color Profile Manager 143
Color Profile Basic Concepts 143

ImageGear Color Profile Groups 144
Color Profile Manager API 145
Working with Global Control Parameters 146
Working with Color Space Conversion Engine 147
Working with Grayscale Look Up Tables 149
Working with Multi-page Images 150
Creating and Deleting a Multi-page Image 150
Opening and Closing an External Image File 151
Loading and Saving Pages 152
Using Other Functions that Work with Pages 155
Using the Multi-page Image Callback Function 156
Working with Image Utility Functions 158
Creating DIBs and DDBs 159
Deleting DIBs and DDBs 159
Reading and Writing Palettes 160
Getting Information about a HIGEAR Image 160
Obtaining Information about Files 161
Working with Library Utility Functions 162
Detecting and Handling Errors 162
Checking the ImageGear Version 164
Working with ImageGear Callback Functions 164
Registering a Callback Function 165
Status Bar Callback 166
Tag (Non-Image) Data Callback 167
Working with Stripped Images 171
Working with Tiled Images 173
Padding 173
Automatic Tile Stitching 176
Saving a TIFF File Using Tiles 177
Working with Run Ends Format 179
Working with ImageGear Digital Camera RAW Filter 185
Using ImageGear Component Manager 187
Component Manager API 187
ImageGear Component Names 188
Using ImageGear ARTX Component 189
Using ImageGear ART Component 189
Loading the ART Component 190
Working with ART Marks 190
ART Component Callback Functions 195

ART Component Important Structures and Types 195
Mark Objects: Descriptions, Types and Attributes 199
ICC_PROFILE_MODE 204

Using ImageGear MD Component 205
Loading and Saving DICOM Images. 205
DICOM Filter Control Parameters. 209
LOAD_CONCAT_REPEATED_DE 209
LOAD_CONVERTTO8G 210
LOAD_DETECTSKIPDIMSE 210
LOAD_MASKALPHACHANNEL 211
LOAD_PAGENUMBER 211
LOAD_SYNTAX 211
LOAD_USE_8x8_LUT 212
LOAD_USE_AUTO_WL_FOR_8G 212
SAVE_ASPART10 213
SAVE_GROUPLENGTHS 213
SAVE_JPEGQUALITY 213
SAVE_LARGEST 214
SAVE_PLANARCONFIG 214
SAVE_SMALLEST 215
SAVE_SYNTAX 215
Associating DICOM Data with an ImageGear Image 216
Reading Data From Data Elements 217
Writing Data to Data Elements 217
Displaying 16-bit Grayscale Images 220
Image Processing of 16-bit Images. 222
Working With DICOM Data Structures 224
Working with DICOM Data Dictionary. 231
Working with Overlays 232
Working with Presentation State Objects 232
Waveform Image Support 234

Using ImageGear PDF Component 234
About the PDF Component. 235
How to Attach the PDF Component to Core ImageGear 237
Working with PDF Layers 239
Distributing PDF and PS Fonts with Your Application. 242
PDF Format Filter 243
PDF Format Filter Control Parameters 244
DEPTH 245
INC_REND 245
INDEPENDENT_PAGESIZE 246
PAGE_HEIGHT 246
PAGE_WIDTH 247
PASSWORD 248
PRINT_DEPTH 248
PRINT_RESOLUTION_X 249
PRINT_RESOLUTION_Y 249
RESOLUTION_X 250
RESOLUTION_Y 250

SAVE_FLAGS . . . 251
SAVE_MAJOR . . . 251
SAVE_MINOR . . . 252
SMOOTH_FLAGS . . . 252
TEXT_ENCODING . . . 253
USE_CROP_BOX . . . 253
PostScript Filter Control Parameters . . . 254
AUTO_POSITION . . . 254
DEPTH . . . 255
HEIGHT . . . 255
INDEPENDENT_PAGESIZE . . . 256
JOB_OPTIONS_FILE . . . 257
LEVEL . . . 257
PAGE_HEIGHT . . . 258
PAGE_WIDTH . . . 258
READ_PREVIEW . . . 259
RESOLUTION_X . . . 260
RESOLUTION_Y . . . 260
SMOOTH_FLAGS . . . 261
TEXT_ENCODING . . . 261
TYPE . . . 262
WIDTH . . . 263
GS PDF Component . . . 263
PDF Format Filter Control Parameters (for HP-UX) . . . 266
DEPTH . . . 267
FILE_NAME . . . 267
GRAPHICS_ALPHA . . . 268
INDEPENDENT_PAGESIZE . . . 269
NOCIE . . . 269
NOTRANSPARENCY . . . 270
PAGE_HEIGHT . . . 270
PAGE_WIDTH . . . 271
RESOLUTION_X . . . 272
RESOLUTION_Y . . . 272
TEXT_ALPHA . . . 273
TEXT_ENCODING . . . 275
USE_CROP_BOX . . . 275
IG_PDF_text_extract . . . 276

Creating Your Imaging Application . . . 279

Compiling and Linking . . . 279
Toolkit Libraries . . . 280
X-Windows System and Math Library . . . 280
Compilation Flags . . . 280

Preparing Your Application for the End User . . . 281

File Format Reference . . . 283

Format Suitability at a Glance . . . 284

ImageGear Support for Graphics File Formats 285
Support for Adobe PDF/PS Formats 286
Support for CAD/Vector Formats 286
Support for DICOM File Format 287
Support for JBIG2 File Format 287
Support for JPEG2000 File Formats 287
Support for Metafile Formats 288
Support for Multimedia File Formats 288
Support for Multi-page File Formats 288
ImageGear Supported Bit Depths 290
ImageGear Supported Compressions Reference 293
ABIC 294
ABIC_BW 295
ABIC_Gray 295
ASCII 295
CCITT Group 3 296
CCITT Group 3 2D 296
CCITT Group 4 297
Deflate 297
FPX NOCHANGE 298
FPX SINCOLOR 298
Huffman 299
IBM MMR 299
JBIG2 300
JPEG 300
JPEG2000 301
Lossless JPEG 303
LZW (Lempel-Ziv-Welch) 303
Packbits 304
Progressive JPEG 304
RAW 305
RLE 305
ImageGear Supported File Formats Reference 306
Adobe DNG 308
Adobe PDF 309
Adobe PS 312
Adobe PSD 316
AFX 320
AVI 320
BMP 322
BTR 325
CAL 326
CGM 327
CLP 329
CUR 331
CUT 332

DCRAW 334
DCX 335
DGN 337
DIB 338
DICOM 340
DWG 345
DXF 347
EPS 349
EXIF-JPEG 352
EXIF-TIFF 355
FlashPix 360
GEM 364
GIF 365
Group 3 367
Group 3 2D 368
Group 4 370
HD Photo 371
HPGL 376
IBM AFP 377
IBM IOCA 379
IBM MO:DCA 381
ICO 383
IFF 385
IMG 387
IMR 388
IMT 389
JBIG2 390
JPEG 391
JPEG2000 402
JPX 403
KFX 405
LV 406
MAC 406
MAC PICT 408
Motion JPEG2000 410
MSP 411
MULTIMEDIA 412
NCR 415
PBM 416
PCD 418
PCX 419
PGM 421
PNG 423
PNM 426
PPM 428
PTOCA 430
QuickTime 431

RAS 432
RAW 434
Scitex CT 436
SGI 439
SVG 441
TGA 442
TIFF 446
TXT (ASCII Text) 458
WBMP 460
WMF 461
WPG 464
XBM 465
XPM 466
XWD 468

ImageGear Alpha Channel Support 470

ImageGear Supported Digital Camera RAW Formats 471

AccuSoft Image Guarantee 477

Identifying the Problem 477

Sending the Image to Accusoft 478

Appendix A: Function Error Return Codes 479

Appendix B: Non-Image Data Storage 493

Metadata Structure “ValueType” and “Value” 493

Non-Image Data Structure 497
TIFF Non-Image Data Structure 497
JPEG Non-Image Data Structure 504
EXIF-TIFF Non-Image Data Structure 512
EXIF-JPEG Non-image Data Structure 519
CIFF Non-Image Data Structure 526
IPTC Non-Image Data Structure 531
PNG Non-Image Data Structure 539
GIF Non-image Data Structure 549
DCRAW Non-Image Data Structure 555

Glossary 559

Bibliography 579

Software License Agreement 581

Schedule A: Additional Terms and Conditions Applicable to Adobe® Technology 586
Schedule B: Additional Terms and Conditions Applicable to MIT's FFTW Technol-

ogy 586
Software Support and Maintenance Policy 588

Volume 2: API Reference 595

Core Component API Function Reference 597

Core Component Functions Reference 597
IG_ascii_import 597
IG_ascii_page_width_get 598
IG_clipboard_copy 599
IG_clipboard_cut 600
IG_clipboard_dimensions 602
IG_clipboard_paste 603
IG_clipboard_paste_available_ex 604
IG_clipboard_paste_merge_ex 605
IG_clipboard_paste_op_get 607
IG_clipboard_paste_op_set 607
IG_close 609
IG_colorspace_conversion_apply_to_image 609
IG_colorspace_conversion_apply_to_raster 610
IG_colorspace_conversion_create 612
IG_colorspace_conversion_create_from_dib_info 614
IG_colorspace_conversion_create_from_image 615
IG_colorspace_conversion_destroy 616
IG_colorspace_conversion_is_applicable 617
IG_colorspace_conversion_result_channels_build 618
IG_colorspace_conversion_result_palette_get 620
IG_color_space_level_get 621
IG_color_space_level_set 622
IG_comm_comp_attach 623
IG_comm_comp_check 624
IG_comm_comp_list 625
IG_comm_entry_request 626
IG_comm_function_call 627
IG_convert_DPI_to_PPM 628
IG_convert_PPM_to_DPI 628
IG_cpm_image_embedded_profile_check 629
IG_cpm_image_profile_get 630
IG_cpm_image_profile_set 632
IG_cpm_profile_get 633
IG_cpm_profile_set 634
IG_cpm_profiles_reset 636
IG_DIB_area_get 636
IG_DIB_area_set 638
IG_DIB_area_size_get 640
IG_DIB_bit_depth_get 642
IG_DIB_channel_count_get 642
IG_DIB_channel_depth_get 643
IG_DIB_channel_depths_get 644
IG_DIB_colorspace_get 645

IG_DIB_column_get 646
IG_DIB_column_set 647
IG_DIB_height_get 649
IG_DIB_info_copy 649
IG_DIB_info_create 650
IG_DIB_info_delete 651
IG_DIB_info_raster_size_get 652
IG_DIB_legacy_bit_depth_get 653
IG_DIB_line_get 654
IG_DIB_line_set 655
IG_DIB_palette_alloc 657
IG_DIB_palette_length_get 658
IG_DIB_palette_pointer_get 659
IG_DIB_palette_size_get 660
IG_DIB_pixel_array_size_get 660
IG_DIB_pixel_get 661
IG_DIB_pixel_set 662
IG_DIB_pix_get 664
IG_DIB_pix_set 665
IG_DIB_raster_get 666
IG_DIB_raster_set 668
IG_DIB_raster_size_get 669
IG_DIB_resolution_get 669
IG_DIB_resolution_set 670
IG_DIB_resolution_units_get 671
IG_DIB_resolution_units_set 672
IG_DIB_row_get 672
IG_DIB_row_set 674
IG_DIB_width_get 675
IG_display_animation_delay_get 676
IG_display_animation_delay_set 676
IG_dspl_antialias_get 677
IG_dspl_antialias_get_ex 678
IG_dspl_antialias_set 679
IG_dspl_antialias_set_ex 680
IG_dspl_artx_image_draw 682
IG_dspl_artx_image_print 683
IG_dspl_artx_page_print 683
IG_dspl_background_get 683
IG_dspl_background_set 684
IG_dspl_DDB_create 685
IG_dspl_DDB_draw 686
IG_dspl_DDB_import 687
IG_dspl_device_to_image 688
IG_dspl_device_to_image_d 690
IG_dspl_dithering_get 691
IG_dspl_dithering_set 691
IG_dspl_foreground_get 692
IG_dspl_foreground_set 693
IG_dspl_free_grp_id_get 694
IG_dspl_gamma_correction_LUT_build 695
IG_dspl_gamma_correction_set 696

IG_dspl_grayscale_LUT_copy_get 698
IG_dspl_grayscale_LUT_exists 698
IG_dspl_grayscale_LUT_update_from 699
IG_dspl_grp_reset 700
IG_dspl_image_calc 701
IG_dspl_image_draw 702
IG_dspl_image_to_device 703
IG_dspl_image_to_device_d 705
IG_dspl_image_wipe 706
IG_dspl_layout_get 708
IG_dspl_layout_set 710
IG_dspl_LUT_get 713
IG_dspl_LUT_set 714
IG_dspl_orientation_get 716
IG_dspl_orientation_set 717
IG_dspl_palette_create 718
IG_dspl_palette_handle 719
IG_dspl_palette_get 720
IG_dspl_palette_set 721
IG_dspl_PPM_correct_get 722
IG_dspl_PPM_correct_set 723
IG_dspl_resize_handle 724
IG_dspl_ROP_get 724
IG_dspl_ROP_set 725
IG_dspl_scroll_get 726
IG_dspl_scroll_handle 727
IG_dspl_scroll_set 729
IG_dspl_scroll_to 731
IG_dspl_scroll_to_ex 732
IG_dspl_transparency_get 733
IG_dspl_transparency_set 734
IG_dspl_update_rect_get 736
IG_dspl_update_rect_set 736
IG_dspl_zoom_get 738
IG_dspl_zoom_set 739
IG_dspl_zoom_to_rect 740
IG_err_callback_get 741
IG_err_callback_set 742
IG_err_count_get 744
IG_err_error_check 744
IG_err_error_get 745
IG_err_error_set 747
IG_err_record_get 748
IG_err_stack_clear 750
IG_error_check 750
IG_error_clear 751
IG_error_get 752
IG_error_set 753
IG_file_IO_register 755
IG_fltr_compressionlist_get 756
IG_fltr_compressionlist_get_ex 758
IG_fltr_ctrl_get 759

IG_fltr_ctrl_list . . . 761
IG_fltr_ctrl_set . . . 762
IG_fltr_detect_FD . . . 763
IG_fltr_detect_file . . . 764
IG_fltr_detect_mem . . . 765
IG_fltr_formatlist_get . . . 766
IG_fltr_formatlist_sort . . . 768
IG_fltr_ICC_callback_get . . . 769
IG_fltr_ICC_callback_set . . . 770
IG_fltr_info_get . . . 771
IG_fltr_load_FD_format . . . 774
IG_fltr_load_file . . . 775
IG_fltr_load_file_format . . . 776
IG_fltr_metad_callback_get . . . 777
IG_fltr_metad_callback_set . . . 778
IG_fltr_metad_update_file . . . 780
IG_fltr_pagecount_FD_format . . . 782
IG_fltr_pagecount_file_format . . . 783
IG_fltr_pagedelete_file . . . 784
IG_fltr_pageinfo_get . . . 785
IG_fltr_pageswap_file . . . 786
IG_fltr_raster_plane_callback_get . . . 787
IG_fltr_raster_plane_callback_set . . . 788
IG_fltr_save_file . . . 789
IG_fltr_savelist_get . . . 790
IG_fltr_savelist_get_ex . . . 792
IG_FX_blur . . . 794
IG_FX_chroma_key . . . 795
IG_FX_diffuse . . . 797
IG_FX_emboss . . . 798
IG_FX_motion . . . 800
IG_FX_noise . . . 801
IG_FX_pixelate . . . 803
IG_FX_posterize . . . 805
IG_FX_spotlight . . . 806
IG_FX_stitch . . . 808
IG_FX_texture . . . 809
IG_FX_twist . . . 811
IG_FX_watermark . . . 812
IG_gctrl_item_by_index_get . . . 813
IG_gctrl_item_count_get . . . 814
IG_gctrl_item_get . . . 815
IG_gctrl_item_id_get . . . 816
IG_gctrl_item_set . . . 816
IG_image_bits_per_channel_get . . . 817
IG_image_blend_with_alpha . . . 818
IG_image_channel_add . . . 819
IG_image_channel_copy_create . . . 820
IG_image_channel_count_get . . . 822
IG_image_channel_depth_get . . . 822
IG_image_channel_depths_get . . . 823
IG_image_channel_depths_change . . . 824

IG_image_channel_remove 825
IG_image_channel_update 826
IG_image_channels_combine 827
IG_image_channels_separate 828
IG_image_colorspace_convert 829
IG_image_colorspace_get 830
IG_image_compression_type_get 831
IG_image_convert 832
IG_image_create 834
IG_image_create_alpha 835
IG_image_create_DIB 837
IG_image_create_DIB_ex 839
IG_image_create_empty 841
IG_image_delete 842
IG_image_DIB_export 843
IG_image_DIB_export_size_calc 845
IG_image_DIB_import 847
IG_image_DIB_info_get 848
IG_image_DIB_palette_pntr_get 849
IG_image_DIB_raster_pntr_get 850
IG_image_dimensions_get 851
IG_image_duplicate 852
IG_image_grayscale_LUT_copy_get 852
IG_image_grayscale_LUT_exists 853
IG_image_grayscale_LUT_update_from 854
IG_image_is_gray 854
IG_image_is_signed_set 855
IG_image_is_valid 856
IG_image_orientation_get 857
IG_image_orientation_set 858
IG_image_resolution_get 859
IG_image_resolution_set 861
IG_info_get_ex 863
IG_info_get_FD_ex 865
IG_info_get_mem_ex 866
IG_initialize 867
IG_IP_alpha_create 868
IG_IP_area_info_get 869
IG_IP_arithmetic 870
IG_IP_arithmetic_rect 871
IG_IP_blend_percent 872
IG_IP_blend_with_alpha 874
IG_IP_blend_with_LUT 877
IG_IP_color_combine_ex 879
IG_IP_color_convert 880
IG_IP_color_count_get 880
IG_IP_color_promote 881
IG_IP_color_reduce_bayer 882
IG_IP_color_reduce_diffuse 883
IG_IP_color_reduce_halftone 884
IG_IP_color_reduce_median_cut 885
IG_IP_color_reduce_octree 886

IG_IP_color_reduce_popularity . . . 888
IG_IP_color_reduce_to_bitonal . . . 889
IG_IP_color_separate . . . 890
IG_IP_contrast_adjust . . . 891
IG_IP_contrast_adjust_ex . . . 893
IG_IP_contrast_equalize . . . 895
IG_IP_contrast_gamma . . . 896
IG_IP_contrast_invert . . . 898
IG_IP_contrast_stretch . . . 900
IG_IP_convert_to_gray . . . 901
IG_IP_convolve_matrix . . . 901
IG_IP_crop . . . 904
IG_IP_decrypt . . . 905
IG_IP_deskew_angle_find . . . 906
IG_IP_deskew_auto . . . 906
IG_IP_despeckle . . . 907
IG_IP_draw_frame . . . 909
IG_IP_edge_map . . . 910
IG_IP_encrypt . . . 911
IG_IP_flip . . . 912
IG_IP_gaussian_blur . . . 913
IG_IP_histo_clear . . . 914
IG_IP_histo_tabulate . . . 914
IG_IP_maximum . . . 916
IG_IP_median . . . 917
IG_IP_merge . . . 918
IG_IP_minimum . . . 920
IG_IP_NR_ROI_control_get . . . 921
IG_IP_NR_ROI_control_set . . . 922
IG_IP_NR_ROI_mask_associate . . . 924
IG_IP_NR_ROI_mask_delete . . . 925
IG_IP_NR_ROI_mask_unassociate . . . 926
IG_IP_NR_ROI_to_HIGEAR_mask . . . 927
IG_IP_pseudocolor_limits . . . 929
IG_IP_pseudocolor_small_grads . . . 931
IG_IP_resize . . . 931
IG_IP_resize_bkgrnd . . . 933
IG_IP_resize_bkgrnd_ex . . . 936
IG_IP_resize_ex . . . 939
IG_IP_RGB_to_hue . . . 942
IG_IP_rotate_any_angle . . . 942
IG_IP_rotate_any_angle_bkgrnd . . . 944
IG_IP_rotate_any_angle_ex . . . 945
IG_IP_rotate_compute_size . . . 946
IG_IP_rotate_multiple_90 . . . 948
IG_IP_sharpen . . . 949
IG_IP_smooth . . . 950
IG_IP_swap_red_blue . . . 951
IG_IP_thumbnail_create . . . 952
IG_IP_thumbnail_create_ex . . . 953
IG_IP_transform_with_LUT . . . 955
IG_IP_unsharp_mask . . . 956

IG_lic_solution_name_set 958
IG_load_alpha_mode_get 958
IG_load_alpha_mode_set 959
IG_load_auto_detect_get 960
IG_load_auto_detect_set 961
IG_load_CCITT_FD 962
IG_load_CCITT_mem 963
IG_load_color_reduction_get 965
IG_load_color_reduction_set 965
IG_load_extra_mode_get 966
IG_load_extra_mode_set 967
IG_load_FD 968
IG_load_FD_CB 970
IG_load_FD_CB_ex 971
IG_load_file 973
IG_load_file_display 974
IG_load_mem 975
IG_load_mem_CB 977
IG_load_mem_CB_ex 978
IG_load_raw_FD 979
IG_load_raw_file 981
IG_load_raw_mem 982
IG_load_rect_get 984
IG_load_rect_set 985
IG_load_size_get 985
IG_load_size_set 986
IG_load_tag_CB_register 987
IG_load_thumbnail 988
IG_load_thumbnail_FD 989
IG_load_thumbnail_mem 990
IG_load_tiles_stitch 991
IG_load_tiles_stitch_FD 993
IG_load_tiles_stitch_mem 995
IG_LUT_copy 997
IG_LUT_copy_to_byte_array 998
IG_LUT_copy_to_word_array 998
IG_LUT_create 999
IG_LUT_destroy 1000
IG_LUT_input_depth_get 1001
IG_LUT_input_is_signed_get 1001
IG_LUT_is_valid 1002
IG_LUT_item_get 1002
IG_LUT_item_set 1003
IG_LUT_length_get 1004
IG_LUT_output_depth_get 1004
IG_LUT_output_is_signed_get 1005
IG_LUT_size_get 1005
IG_LUT_update_from_byte_array 1006
IG_LUT_update_from_word_array 1007
IG_mem_CB_register 1007
IG_mpf_info_get 1009
IG_mpf_page_count_get 1010

IG_mpf_page_delete 1011
IG_mpf_page_get 1012
IG_mpf_page_info_get 1013
IG_mpf_page_info_get_ex 1015
IG_mpf_page_load 1016
IG_mpf_page_save 1017
IG_mpf_page_swap 1019
IG_mpf_page_unload 1020
IG_mpf_tile_count_get 1021
IG_mpi_CB_get 1022
IG_mpi_CB_reset 1024
IG_mpi_CB_reset_all 1025
IG_mpi_CB_set 1027
IG_mpi_close 1030
IG_mpi_create 1031
IG_mpi_delete 1032
IG_mpi_file_open 1032
IG_mpi_file_save 1034
IG_mpi_info_get 1035
IG_mpi_is_valid 1037
IG_mpi_page_count_get 1038
IG_mpi_page_count_set 1039
IG_mpi_page_delete 1041
IG_mpi_page_get 1042
IG_mpi_page_is_valid 1043
IG_mpi_page_set 1043
IG_page_count_get 1045
IG_page_count_get_FD 1046
IG_page_count_get_mem 1046
IG_palette_entry_get 1047
IG_palette_entry_set 1048
IG_palette_get 1049
IG_palette_load 1050
IG_palette_save 1051
IG_palette_set 1053
IG_pixel_bits_per_channel_get 1054
IG_pixel_channel_count_get 1054
IG_pixel_create 1055
IG_pixel_data_pointer_get 1056
IG_pixel_delete 1057
IG_pixel_value_get 1058
IG_pixel_value_set 1059
IG_runs_row_get 1061
IG_runs_row_set 1063
IG_save_FD 1064
IG_save_FD_CB_ex 1066
IG_save_file 1068
IG_save_file_size_calc 1069
IG_save_JPEG_quality_get 1070
IG_save_JPEG_quality_set 1071
IG_save_mem 1072
IG_save_mem_CB_direct 1075

IG_save_mem_CB_ex 1076
IG_save_tag_CB_register 1078
IG_save_thumbnail_set 1079
IG_status_bar_CB_register 1080
IG_thread_data_ID_associate 1082
IG_thread_data_ID_get 1083
IG_thread_local_data_cleanup 1084
IG_thread_image_lock 1084
IG_thread_image_unlock 1086
IG_tile_count_get 1087
IG_tile_count_get_FD 1088
IG_tile_count_get_mem 1089
IG_util_colorspace_alpha_count_get 1090
IG_util_colorspace_color_count_get 1090
IG_util_colorspace_contains_alpha 1091
IG_util_colorspace_contains_extra 1092
IG_util_colorspace_extra_count_get 1092
IG_util_colorspace_is_premultiplied 1093
IG_util_colorspace_is_valid 1094
IG_util_colorspace_value_to_ids 1095
IG_util_version_get 1095
IG_version_compile_date 1096
IG_version_is_eval 1097
IG_version_numbers 1097

Core Component Callback Functions Reference 1098
LPAFT_IG_ICC_GET_CB 1099
LPAFT_IG_METAD_ITEM_ADD_CB 1099
LPAFT_IG_METAD_ITEM_GET_CB 1101
LPAFT_IG_METAD_ITEM_SET_CB 1103
LPFNIG_DIB_CREATE 1105
LPFNIG_DIB_CREATE_EX 1106
LPFNIG_DIB_GET 1106
LPFNIG_DIB_GET_EX 1107
LPFNIG_LOAD_DISP 1108
LPFNIG_MEM_ALLOC 1109
LPFNIG_MEM_FREE 1110
LPFNIG_MEM_REALLOC 1110
LPFNIG_MPCB_UPDATE 1111
LPFNIG_RASTER_GET 1112
LPFNIG_RASTER_SET 1113
LPFNIG_READ 1114
LPFNIG_SEEK 1115
LPFNIG_SIZE_CHANGE 1116
LPFNIG_STATUS_BAR 1116
LPFNIG_TAG_GET 1118
LPFNIG_TAG_SET 1121
LPFNIG_TAG_USER_GET 1125
LPFNIG_WRITE 1127

ART Component API Function Reference 1129

ART Component Functions Reference 1129

ART_callback_register 1131
ART_control_option_get 1132
ART_control_option_set 1133
ART_edit_copy 1135
ART_edit_cut 1137
ART_edit_paste 1138
ART_edit_paste_count 1140
ART_edit_undo 1140
ART_edit_undo_clear 1141
ART_edit_undo_count 1142
ART_edit_undo_enable 1143
ART_edit_undo_record 1143
ART_enable_rotation_get 1145
ART_enable_rotation_set 1145
ART_file_append 1146
ART_file_export 1147
ART_file_import 1148
ART_file_import_ex 1150
ART_global_block_create 1151
ART_global_block_delete 1152
ART_global_block_first 1153
ART_global_block_next 1154
ART_global_block_query 1156
ART_group_access 1157
ART_group_bring_to_front 1158
ART_group_delete 1159
ART_group_edit_copy 1160
ART_group_edit_cut 1161
ART_group_select 1162
ART_group_send_to_back 1163
ART_group_visible 1163
ART_image_burn_in 1165
ART_mark_block_create 1166
ART_mark_block_create_ex 1168
ART_mark_block_delete 1170
ART_mark_block_first 1171
ART_mark_block_next 1172
ART_mark_block_query 1174
ART_mark_bring_to_front 1175
ART_mark_count 1176
ART_mark_create 1177
ART_mark_delete 1178
ART_mark_first 1179
ART_mark_is_selected 1181
ART_mark_is_visible 1182
ART_mark_modify 1182
ART_mark_next 1184
ART_mark_query 1185
ART_mark_rotate 1186

ART_mark_select 1187
ART_mark_select_count 1188
ART_mark_selected_first 1189
ART_mark_selected_next 1190
ART_mark_send_to_back 1190
ART_mark_visible 1191
ART_rotation_angle_get 1193
ART_rotation_angle_set 1193
ART_version_numbers 1194
ART Component Macros 1195
ART_UNIX_LEGACY_FONTS_LOAD_GET 1195
ART_UNIX_LEGACY_FONTS_LOAD_SET 1196
ART Component Callback Functions Reference 1196
LPFNART_GUI_TEXTREPLACEFUNC 1197
LPFNART_GUI_TOOLBARNOTIFYFUNC 1198
LPFNIG_ART_CREATENOTIFYFUNC 1199
LPFNIG_ART_DELETENOTIFYFUNC 1200
LPFNIG_ART_DRAWNOTIFYFUNC 1201
LPFNIG_ART_MODIFYNOTIFYFUNC 1203
LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC 1205
LPFNIG_ART_PREMODIFYNOTIFYFUNC 1207
LPFNIG_ART_SELECTNOTIFYFUNC 1208
LPFNIG_ART_TEXTOVERFLOWNOTIFYFUNC 1210

MD Component API Function Reference 1213

MED_DCM_DS_bits_get 1213
MED_DCM_DS_create 1215
MED_DCM_DS_curr_data_get 1216
MED_DCM_DS_curr_data_get_string 1217
MED_DCM_DS_curr_data_set 1218
MED_DCM_DS_curr_index_get 1219
MED_DCM_DS_curr_info_get 1219
MED_DCM_DS_curr_remove 1220
MED_DCM_DS_DE_insert 1221
MED_DCM_DS_destroy 1222
MED_DCM_DS_exists 1223
MED_DCM_DS_info_get 1224
MED_DCM_DS_is_empty 1224
MED_DCM_DS_LUT_copy_get 1225
MED_DCM_DS_LUT_exists 1226
MED_DCM_DS_LUT_update_from 1227
MED_DCM_DS_move_ascend 1228
MED_DCM_DS_move_descend 1229
MED_DCM_DS_move_find 1230
MED_DCM_DS_move_find_first 1232
MED_DCM_DS_move_first 1234
MED_DCM_DS_move_index 1236
MED_DCM_DS_move_last 1237
MED_DCM_DS_move_next 1239
MED_DCM_DS_move_prev 1241
MED_DCM_DS_orig_TS_get 1243

MED_DCM_DS_part10_get 1244
MED_DCM_DS_part10_set 1245
MED_DCM_DS_PixPadVal_get 1245
MED_DCM_DS_PixPadVal_set 1247
MED_DCM_DS_preamble_get 1248
MED_DCM_DS_preamble_set 1249
MED_DCM_DS_Rescale_get 1250
MED_DCM_DS_TAG_ELEMENT 1251
MED_DCM_DS_TAG_GROUP 1251
MED_DCM_DS_TAG_MAKE 1252
MED_DCM_DS_TS_get 1253
MED_DCM_DS_TS_set 1254
MED_DCM_DS_update_file 1254
MED_DCM_DS_Window_Level_get 1255
MED_DCM_load_DICOM 1256
MED_DCM_load_DICOM_FD 1257
MED_DCM_save_DICOM 1258
MED_DCM_save_DICOM_FD 1261
MED_DCM_util_data_to_string 1263
MED_DCM_util_tag_info_add 1264
MED_DCM_util_tag_info_free 1265
MED_DCM_util_tag_info_get 1265
MED_DCM_util_VR_info_mode 1266
MED_DCM_util_VR_info_string 1268
MED_display_color_create 1269
MED_display_color_limits 1271
MED_display_color_set 1273
MED_display_contrast 1274
MED_display_contrast_auto 1275
MED_display_grayscale_LUT_build 1277
MED_display_grayscale_LUT_build_auto 1278
MED_IP_contrast 1278
MED_IP_contrast_auto 1279
MED_IP_high_bit_transform 1280
MED_IP_histo_clear 1281
MED_IP_histo_tabulate 1282
MED_IP_min_max 1283
MED_IP_normalize 1284
MED_IP_promote_to_16_gray 1285
MED_IP_reduce_depth_with_downshift 1285
MED_IP_reduce_depth_with_LUT 1286
MED_IP_swap_bytes 1287
MED_modality_transform_apply 1288
MED_overlay_apply 1288
MED_overlay_extract 1289
MED_PS_apply 1290
MED_PS_display_contrast 1291
MED_PS_display_contrast_auto 1292
MED_PS_extract 1293
MED_PS_GSDF_LUT_build 1294
MED_PS_GSDF_LUT_init 1295
MED_PS_pres_LUT_get 1296

MED_PS_pres_LUT_info_get 1297
MED_PS_pres_LUT_set 1297
MED_PS_pres_state_GSDF_apply 1298
MED_VOI_window_init_from_min_max 1299
MED_VOI_window_max_get 1299
MED_VOI_window_min_get 1300
AT_MED_DCM_DISPLAY_SETTINGS 1300
AT_MED_MODALITY_RESCALE 1301
AT_MED_PIXEL_PADDING_SETTINGS 1302
AT_MED_VOI_WINDOW 1302

. . . . . PDF Component API Function Reference 1305

Summary 1305

General API 1311
C API 1312
IG_PDE_get_default_gstate 1314
IG_PDF_get_host_encoding 1314
IG_PDF_initialize 1315
IG_PDF_register_authproc 1317
IG_PDF_terminate 1317
IG_PDF_text_extract 1318
IG_PDF_translate_to_host 1318
IG_PDF_translate_to_pdf 1319
AM_PDF_LONG_TO_FIXED 1320
AM_PDF_FIXED_ROUND_TO_LONG 1321
AM_PDF_FIXED_TRUNC_TO_LONG 1321
AM_PDF_SHORT_TO_FIXED 1322
AM_PDF_FIXED_ROUND_TO_SHORT 1322
AM_PDF_FIXED_TRUNC_TO_SHORT 1323
AM_PDF_DOUBLE_TO_FIXED 1323
AM_PDF_FIXED_TO_DOUBLE 1324
LPFNIG_PDF_AUTHPROC 1324
LPFNIG_PDF_STREAM_PROC 1325
LPFNIG_PDF_STREAM_DESTROYPROC 1326
LPFNIG_PDF_SYSFONT_ENUMPROC 1327
LPFNIG_PDE_CLIP_ENUMPROC 1327

Basic Objects 1328
HIG_PDF_BASOBJ 1330
IG_PDF_basobj_get_type 1330
IG_PDF_basobj_release 1330
IG_PDF_basobj_remove 1331
HIG_PDF_BASBOOL 1332
IG_PDF_basbool_create 1332
IG_PDF_basbool_get_value 1333
HIG_PDF_BASINT 1334
IG_PDF_basint_create 1334
IG_PDF_basint_get_value 1335
HIG_PDF_BASFIXED 1336
IG_PDF_basfixed_create 1336

IG_PDF_basfixed_get_value 1337
HIG_PDF_BASSTR 1338
IG_PDF_basstr_create 1338
IG_PDF_basstr_get_value 1339
HIG_PDF_BASNAME 1340
IG_PDF_basname_create 1340
IG_PDF_basname_get_value 1341
HIG_PDF_BASARR 1342
IG_PDF_basarr_create 1342
IG_PDF_basarr_get_length 1343
IG_PDF_basarr_get 1344
IG_PDF_basarr_put 1344
IG_PDF_basarr_put_int 1345
IG_PDF_basarr_put_fixed 1346
IG_PDF_basarr_put_bool 1346
IG_PDF_basarr_put_name 1347
IG_PDF_basarr_remove 1348
IG_PDF_basarr_remove_nth 1348
HIG_PDF_BASDICT 1349
IG_PDF_basdict_create 1349
IG_PDF_basdict_known 1350
IG_PDF_basdict_get 1351
IG_PDF_basdict_put 1351
IG_PDF_basdict_put_int 1352
IG_PDF_basdict_put_fixed 1353
IG_PDF_basdict_put_bool 1353
IG_PDF_basdict_put_name 1354
IG_PDF_basdict_remove 1355
HIG_PDF_BASNULL 1356
IG_PDF_basnull_create 1356

General Objects 1356
HIG_PDF_ACTION 1358
IG_PDF_action_create 1358
IG_PDF_action_create_destination 1359
IG_PDF_action_create_filename 1360
IG_PDF_action_delete 1360
IG_PDF_action_get_destination 1361
IG_PDF_action_get_dictionary 1361
IG_PDF_action_get_filename 1362
IG_PDF_action_get_type 1363
HIG_PDF_ATOM 1364
IG_PDF_atom_from_string 1364
IG_PDF_atom_get_string 1365
HIG_PDF_BOOKMARK 1366
IG_PDF_bookmark_add_child 1368
IG_PDF_bookmark_add_new_child 1368
IG_PDF_bookmark_add_next 1369
IG_PDF_bookmark_add_new_sibling 1369
IG_PDF_bookmark_add_prev 1370
IG_PDF_bookmark_add_subtree 1371
IG_PDF_bookmark_delete 1371

IG_PDF_bookmark_find_title 1372
IG_PDF_bookmark_get_action 1372
IG_PDF_bookmark_get_color 1373
IG_PDF_bookmark_get_count 1373
IG_PDF_bookmark_get_first_child 1374
IG_PDF_bookmark_get_flags 1374
IG_PDF_bookmark_get_indent 1375
IG_PDF_bookmark_get_last_child 1375
IG_PDF_bookmark_get_next 1376
IG_PDF_bookmark_get_parent 1376
IG_PDF_bookmark_get_prev 1377
IG_PDF_bookmark_get_title 1378
IG_PDF_bookmark_has_children 1378
IG_PDF_bookmark_is_open 1379
IG_PDF_bookmark_remove 1379
IG_PDF_bookmark_remove_action 1380
IG_PDF_bookmark_set_action 1380
IG_PDF_bookmark_set_color 1381
IG_PDF_bookmark_set_flags 1381
IG_PDF_bookmark_set_open 1382
IG_PDF_bookmark_set_title 1382
IG_PDF_bookmark_unlink 1383
HIG_PDF_STREAM 1384
IG_PDF_stream_open_mem_for_read 1384
IG_PDF_stream_read_CB_register 1385
IG_PDF_stream_write_CB_register 1386
IG_PDF_stream_read 1386
IG_PDF_stream_flush 1387
IG_PDF_stream_close 1388
HIG_PDF_DESTINATION 1389
IG_PDF_destination_create 1389
IG_PDF_destination_delete 1390
IG_PDF_destination_get_explicit_attrs 1391
IG_PDF_destination_get_named_attrs 1391
IG_PDF_destination_get_type 1392
IG_PDF_destination_remove 1393
IG_PDF_destination_resolve 1393
HIG_PDF_DICTIONARY 1395
IG_PDF_dictionary_create 1395
IG_PDF_dictionary_get_layer 1396
IG_PDF_dictionary_get_layer_count 1397
IG_PDF_dictionary_get_unique_id 1397
IG_PDF_dictionary_get_vis_policy 1398
IG_PDF_dictionary_release 1398
HIG_PDF_DOC 1399
IG_PDF_doc_create 1401
IG_PDF_doc_create_new_page 1401
IG_PDF_doc_delete_pages 1402
IG_PDF_doc_insert_pages 1403
IG_PDF_doc_get_bookmark 1404
IG_PDF_doc_get_info 1405

IG_PDF_doc_get_layer 1406
IG_PDF_doc_get_layer_count 1406
IG_PDF_doc_get_page_count 1407
IG_PDF_doc_set_info 1407
IG_PDF_doc_print 1408
IG_PDF_doc_create_wordfinder 1409
IG_PDF_doc_create_wordfinder_ucs 1410
IG_PDF_doc_get_new_crypt_handler 1411
IG_PDF_doc_get_new_security_data 1412
IG_PDF_doc_get_new_security_info 1413
IG_PDF_doc_get_security_data 1413
IG_PDF_doc_perm_request 1414
IG_PDF_doc_set_new_crypt_handler 1415
IG_PDF_doc_set_new_security_data 1416
HIG_PDF_FONT 1417
IG_PDF_font_get_bbox 1418
IG_PDF_font_get_charset 1419
IG_PDF_font_get_cid_systeminfo 1419
IG_PDF_font_get_cid_system_supplement 1420
IG_PDF_font_get_descendant 1420
IG_PDF_font_get_encoding_index 1421
IG_PDF_font_get_encoding_name 1421
IG_PDF_font_get_font_matrix 1422
IG_PDF_font_get_metrics 1422
IG_PDF_font_get_name 1423
IG_PDF_font_get_subtype 1423
IG_PDF_font_get_widths 1424
IG_PDF_font_is_embedded 1424
IG_PDF_font_set_metrics 1425
IG_PDF_font_translate_string 1425
IG_PDF_font_translate_to_host 1426
IG_PDF_font_translate_to_ucs 1427
IG_PDF_font_translate_widths 1428
HIG_PDF_LAYER 1429
IG_PDF_layer_create 1430
IG_PDF_layer_get_name 1430
IG_PDF_layer_set_name 1431
IG_PDF_layer_get_current_state 1432
IG_PDF_layer_set_current_state 1432
IG_PDF_layer_get_initial_state 1433
IG_PDF_layer_set_initial_state 1433
IG_PDF_layer_has_usage_info 1434
IG_PDF_layer_get_usage_info 1434
IG_PDF_layer_set_usage_info 1435
IG_PDF_layer_get_intent 1436
IG_PDF_layer_set_intent 1437
IG_PDF_layer_get_unique_id 1438
IG_PDF_layer_delete 1438
IG_PDF_layer_release 1439
IG_PDF_layer_remove 1439
HIG_PDF_PAGE 1441
IG_PDF_page_get_content 1441

IG_PDF_page_get_crop_box 1442
IG_PDF_page_get_rotation 1442
IG_PDF_page_make_color_separations 1443
IG_PDF_page_set_content 1444
IG_PDF_page_release_content 1444
IG_PDF_page_get_annotation_count 1445
HIG_PDF_WORDFINDER 1446
IG_PDF_wordfinder_acquire_wordlist 1446
IG_PDF_wordfinder_release_wordlist 1447
IG_PDF_wordfinder_get_word 1447
IG_PDF_wordfinder_delete 1448
HIG_PDF_WORD 1449
IG_PDF_word_get_char_offset 1449
IG_PDF_word_get_char_style 1450
IG_PDF_word_get_length 1451
IG_PDF_word_get_quad 1451
IG_PDF_word_get_quad_count 1452
IG_PDF_word_get_string 1452
IG_PDF_word_get_style_transition 1453
IG_PDF_word_delete 1454
HIG_PDF_STYLE 1455
IG_PDF_style_get_color 1455
IG_PDF_style_get_font 1455
IG_PDF_style_get_font_size 1456
IG_PDF_style_delete 1456
HIG_PDF_SYSFONT 1458
IG_PDF_sysfont_enumerate 1458
IG_PDF_sysfont_find 1459
IG_PDF_sysfont_find_for_pdefont 1460
IG_PDF_sysfont_get_platform_data 1460
IG_PDF_sysfont_get_attrs 1461
IG_PDF_sysfont_get_cid_system_info 1462
IG_PDF_sysfont_get_create_flags 1462
IG_PDF_sysfont_get_info 1463
IG_PDF_sysfont_get_name 1464
IG_PDF_sysfont_get_widths 1464
IG_PDF_sysfont_release_platform_data 1465
IG_PDF_sysfont_release 1465
HIG_PDF_SYSENCODING 1466
IG_PDF_sysencoding_create_from_base_name 1466
IG_PDF_sysencoding_create_from_cmap_name 1467
IG_PDF_sysencoding_create_from_code_page 1467
IG_PDF_sysencoding_get_writing_mode 1468
IG_PDF_sysencoding_is_identity 1469
IG_PDF_sysencoding_is_multibyte 1469
IG_PDF_sysencoding_release 1470
Page Editing Objects and Elements 1470
HIG_PDE_OBJECT 1474
IG_PDE_object_get_type 1474
HIG_PDE_CONTENT 1475

IG_PDE_content_create 1475
IG_PDE_content_get_element 1476
IG_PDE_content_add_element 1476
IG_PDE_content_remove_element 1477
IG_PDE_content_get_element_count 1477
IG_PDE_content_get_default_color_space 1478
IG_PDE_content_get_attrs 1478
HIG_PDE_COLORSPACE 1480
IG_PDE_colorspace_create 1480
IG_PDE_colorspace_get_base_name 1481
IG_PDE_colorspace_get_base_color_components 1482
IG_PDE_colorspace_get_ctable 1482
IG_PDE_colorspace_get_hival 1483
IG_PDE_colorspace_get_name 1484
IG_PDE_colorspace_get_color_components 1484
IG_PDE_colorspace_release 1485
HIG_PDE_FONT 1486
IG_PDE_font_create 1488
IG_PDE_font_create_from_sysfont 1490
IG_PDE_font_create_from_sysfont_and_encoding 1491
IG_PDE_font_create_from_sysfont_with_params 1491
IG_PDE_font_get_attrs 1492
IG_PDE_font_create_tounicode_now 1492
IG_PDE_font_create_widths_now 1493
IG_PDE_font_embed_now 1494
IG_PDE_font_embed_now_dont_subset 1494
IG_PDE_font_get_create_need_flags 1495
IG_PDE_font_get_codebyte_count 1495
IG_PDE_font_get_onebyte_encoding 1496
IG_PDE_font_get_sysencoding 1497
IG_PDE_font_get_sysfont 1497
IG_PDE_font_get_widths 1498
IG_PDE_font_get_widths_now 1498
IG_PDE_font_is_embedded 1499
IG_PDE_font_is_multibyte 1499
IG_PDE_font_set_sysencoding 1500
IG_PDE_font_set_sysfont 1500
IG_PDE_font_subset_now 1501
IG_PDE_font_sum_widths 1501
IG_PDE_font_translate_glyphids_to_unicode 1502
IG_PDE_font_release 1503
HIG_PDE_ELEMENT 1504
IG_PDE_element_get_type 1504
IG_PDE_element_clone 1505
IG_PDE_element_is_at_point 1505
IG_PDE_element_is_at_rect 1506
IG_PDE_element_get_bbox 1507
IG_PDE_element_get_clip 1508
IG_PDE_element_set_clip 1508
IG_PDE_element_get_gstate 1509
IG_PDE_element_set_gstate 1509
IG_PDE_element_get_matrix 1510

IG_PDE_element_set_matrix 1511
IG_PDE_element_get_dictionary 1511
IG_PDE_element_get_unique_id 1512
IG_PDE_element_has_gstate 1512
IG_PDE_element_release 1513
HIG_PDE_CLIP 1514
IG_PDE_clip_create 1514
IG_PDE_clip_clone 1515
IG_PDE_clip_add_element 1515
IG_PDE_clip_get_element 1516
IG_PDE_clip_remove_elements 1517
IG_PDE_clip_get_element_count 1517
IG_PDE_clip_enumerate_elements 1518
HIG_PDE_CONTAINER 1520
IG_PDE_container_create 1520
IG_PDE_container_get_content 1521
IG_PDE_container_set_content 1521
IG_PDE_container_get_dictionary 1522
IG_PDE_container_set_dictionary 1523
IG_PDE_container_get_mctag 1523
IG_PDE_container_set_mctag 1524
HIG_PDE_FORM 1525
IG_PDE_form_create 1525
IG_PDE_form_clone 1526
IG_PDE_form_get_content 1527
IG_PDE_form_set_content 1527
IG_PDE_form_has_xgroup 1528
IG_PDE_form_get_xgroup 1528
IG_PDE_form_set_xgroup 1529
HIG_PDE_GROUP 1530
IG_PDE_group_create 1530
IG_PDE_group_get_content 1531
IG_PDE_group_set_content 1531
HIG_PDE_IMAGE 1532
IG_PDE_image_create 1533
IG_PDE_image_is_data_encoded 1535
IG_PDE_image_get_attrs 1535
IG_PDE_image_get_color_mask 1536
IG_PDE_image_get_colorspace 1537
IG_PDE_image_get_data 1537
IG_PDE_image_get_data_length 1538
IG_PDE_image_get_data_stream 1539
IG_PDE_image_get_decode_array 1539
IG_PDE_image_get_filter_array 1540
IG_PDE_image_get_matte_array 1541
IG_PDE_image_get_soft_mask 1541
IG_PDE_image_has_soft_mask 1542
IG_PDE_image_is_xobject 1542
IG_PDE_image_set_color_mask 1543
IG_PDE_image_set_colorspace 1544
IG_PDE_image_set_data 1544

IG_PDE_image_set_data_stream 1545
IG_PDE_image_set_decode_array 1546
IG_PDE_image_set_matte_array 1547
IG_PDE_image_set_soft_mask 1547
HIG_PDE_PATH 1548
IG_PDE_path_create 1548
IG_PDE_path_add_segment 1549
IG_PDE_path_get_data 1550
IG_PDE_path_set_data 1551
IG_PDE_path_get_paint_op 1551
IG_PDE_path_set_paint_op 1552
HIG_PDE_PLACE 1553
IG_PDE_place_create 1553
IG_PDE_place_get_dictionary 1554
IG_PDE_place_set_dictionary 1555
IG_PDE_place_get_mctag 1555
IG_PDE_place_set_mctag 1556
HIG_PDE_POSTSCRIPT 1557
IG_PDE_postscript_create 1557
IG_PDE_postscript_get_attrs 1558
IG_PDE_postscript_get_data 1559
IG_PDE_postscript_set_data 1559
IG_PDE_postscript_get_data_stream 1560
IG_PDE_postscript_set_data_stream 1560
HIG_PDE_SOFTMASK 1562
IG_PDE_softmask_create 1562
IG_PDE_softmask_create_from_name 1563
IG_PDE_softmask_get_form 1564
IG_PDE_softmask_set_form 1564
IG_PDE_softmask_get_backdrop_color 1565
IG_PDE_softmask_set_backdrop_color 1565
IG_PDE_softmask_get_name 1566
HIG_PDE_SHADING 1567
IG_PDE_shading_create 1567
HIG_PDE_TEXT 1568
IG_PDE_text_create 1570
IG_PDE_text_add 1570
IG_PDE_text_add_item 1571
IG_PDE_text_get_advance 1572
IG_PDE_text_get_bbox 1573
IG_PDE_text_get_font 1574
IG_PDE_text_get_gstate 1575
IG_PDE_text_get_item 1576
IG_PDE_text_get_matrix 1577
IG_PDE_text_get_byte_count 1578
IG_PDE_text_get_char_count 1578
IG_PDE_text_get_runs_count 1579
IG_PDE_text_get_quad 1579
IG_PDE_text_get_run_for_char 1580
IG_PDE_text_get_state 1581
IG_PDE_text_get_stroke_matrix 1582
IG_PDE_text_get_text 1582

IG_PDE_text_is_at_point 1583
IG_PDE_text_is_at_rect 1584
IG_PDE_text_remove 1585
IG_PDE_text_remove_items 1586
IG_PDE_text_replace_chars 1587
IG_PDE_text_run_get_char_offset 1587
IG_PDE_text_run_get_char_count 1588
IG_PDE_text_run_set_font 1588
IG_PDE_text_run_set_gstate 1589
IG_PDE_text_run_set_matrix 1590
IG_PDE_text_run_set_state 1590
IG_PDE_text_run_set_stroke_matrix 1591
IG_PDE_text_split_run_at 1591
HIG_PDE_TEXTITEM 1593
IG_PDE_textitem_create 1593
IG_PDE_textitem_copy_text 1594
IG_PDE_textitem_get_font 1595
IG_PDE_textitem_set_font 1595
IG_PDE_textitem_get_gstate 1596
IG_PDE_textitem_set_gstate 1597
IG_PDE_textitem_get_text_length 1597
IG_PDE_textitem_get_matrix 1598
IG_PDE_textitem_set_matrix 1598
IG_PDE_textitem_get_state 1599
IG_PDE_textitem_set_state 1599
IG_PDE_textitem_remove_chars 1600
IG_PDE_textitem_replace_chars 1600
IG_PDE_textitem_replace_text 1601
HIG_PDE_XGROUP 1602
IG_PDE_xgroup_create 1602
IG_PDE_xgroup_get_colorspace 1603
IG_PDE_xgroup_set_colorspace 1603
IG_PDE_xgroup_get_isolated 1604
IG_PDE_xgroup_set_isolated 1604
IG_PDE_xgroup_get_knockout 1605
IG_PDE_xgroup_set_knockout 1605
HIG_PDE_XOBJECT 1607
IG_PDE_xobject_create 1607

Subsidiary Types / DLL Structures 1608
AT_PDF_FIXEDPOINT 1609
AT_PDF_FIXEDMATRIX 1610
AT_PDF_FIXEDRECT 1611
AT_PDF_FIXEDQUAD 1611
AT_PDF_FONT_STYLES 1612
AT_PDF_FONT_METRICS 1613
AT_PDF_PRINTOPTIONS 1614
AT_PDF_PRINTPARAMS 1615
AT_PDF_SECURITYDATA 1622
AT_PDE_FONTATTRS 1624
AT_PDE_COLORVALUE 1626
AT_PDE_COLORSPEC 1627

AT_PDE_DASH 1627
AT_PDE_GRAPHICSTATE 1628
AT_PDE_TEXTSTATE 1629
AT_PDF_SYSFONT_PLATDATA 1631
AT_PDE_FONT_INFO 1632
AT_PDE_FONT_CREATEFROMSYSFONTPARAMS 1633
AT_PDE_COLORRANGE 1634
AT_PDE_XYZCOLOR 1635
AT_PDE_COLORDATA_CALGRAY 1635
AT_PDE_COLORDATA_CALRGB 1636
AT_PDE_COLORDATA_LAB 1637
AT_PDE_COLORDATA_ICCBASED 1637
AT_PDE_COLORDATA_INDEXED 1638
AT_PDE_COLORDATA_SEPARATION 1639
AT_PDE_COLORDATA_DEVICEN 1639
AT_PDE_COLORDATA 1640
AT_PDE_PSATTRS 1641
AT_PDE_IMAGEATTRS 1642
AT_PDE_FILTERSPEC 1643
AT_PDE_FILTERARRAY 1643
AT_PDE_CONTENTATTRS 1644

Enumerations 1645
enumIGPDFBookmarkFlags 1645
enumIGPDFCharset 1646
enumIGPDFDestinationType 1646
enumIGPDFFixedValues 1646
enumIGPDEType 1648
enumIGPDEFontProtection 1650
enumIGPDETextFlags 1650
enumIGPDFPageNumber 1651
enumIGPDEInsertElement 1651
enumIGPDFFontFlags 1652
enumIGPDEFontCreateFlags 1652
enumIGPDEFontCreateNeedFlags 1653
enumIGPDFSysFontPackageType 1654
enumIGPDFSysFontMatchFlags 1654
enumIGPDEGraphicStateWasSetFlags 1655
enumIGPDEImageDataFlags 1656
enumIGPDEXGroupCreateFlags 1657
enumIGPDESoftMaskCreateFlags 1657
enumIGPDEImageAttrFlags 1657
enumIGPDEPSAttrFlags 1658
enumIGPDEElementCopyFlags 1658
enumIGPDEContentGetResourceFlags 1659
enumIGPDEContentFlags 1659
enumIGPDFWordFlags 1660
enumIGPDFWordFinderVersion 1660
enumIGPDFCodePages 1661
enumIGPDEPathElementType 1662
enumIGPDEPathOpFlags 1663
enumIGPDFBasicType 1664

enumIGPDFPermsFlags 1664
enumIGPDFRevision 1666
enumIGPDFSecurityInfoFlags 1667
enumIGPDFStdSecurityMethod 1667
GS PDF Component API 1668
IG_PDF_text_extract 1668

Volume 1: User Guide

The first Volume of the ImageGear User's Manual provides you with the following information:

- The "Introduction" Chapter provides information about ImageGear features, improvements, component structure, plus information about AccuSoft's Product Support Services and descriptions of AccuSoft's other products.
- The "Installation" Chapter provides system requirements information as well as installation instructions. It also describes the structure of installed ImageGear files and the ImageGear Licensing process.
- The "Getting Started" Chapter describes how to incorporate ImageGear's imaging capabilities into your own applications by using the sample source imaging programs provided with your toolkit.
- The "Understanding Bitmap Images" Chapter explains Device-Independent Bitmaps (DIBs), the memory structures that are at the heart of ImageGear's architecture. The majority of ImageGear functions operate either directly or indirectly upon DIBs. Device-Dependent Bitmaps (DDBs) are also discussed, as is the meaning of Pixels (the building blocks of bitmaps), and all color and gray-scale representations supported by ImageGear.
- The "Using ImageGear" Chapter provides information about how to call an ImageGear function, including setting up the variables and structures passed as arguments, and details which functions to use to perform the variety of imaging tasks. It also explains how to work with some ImageGear components API.
- The Creating Your Imaging Application Chapter guides you through the steps of creating, testing, and finalizing your application. This discussion includes declaring compile/link switches and default directories, activating debug mode, and interpreting the function success/error return codes.
- The "File Format Reference" Chapter provides detailed information on each of the file formats supported by ImageGear (see the section ImageGear Supported File Formats), its control parameters, supported features and compressions, plus "at-a-glance" tables that describe the features supported by each format and provide guidelines on which formats may be most suitable when speed, storage space, or similar considerations are important.
- The AccuSoft Image Guarantee provides information about how to identify an imaging problem, if it occurs.

Besides this there is some supplemental information:

- See "Appendix A: Function Error Return Codes" for lists of error codes returned by ImageGear functions. These error codes are classified into different groups and numbered in alphabetical and numerical orders.
- See "Appendix B: Non-Image Data Storage" for detailed info about TIFF, JPEG, EXIF-TIFF, EXIF-JPEG, IPTC, PNG and GIF formats non-image data storage.
- Software License Agreement provides the AccuSoft Software License Agree-

ment and Software Support and Maintenance Policy documents.

This Volume also provides a Glossary and Bibliography.

Introduction

ImageGear is the most advanced way to create, control, and deliver more secure, high-quality imaging applications. Using ImageGear, you can easily add powerful imaging capability to your applications. ImageGear supports all of the most commonly used graphics file formats (and many more), providing complete compatibility when developing applications across multiple platforms, or when developing for users who have a variety of target systems.

ImageGear features include analysis, correction, and manipulation of both computer-generated and scanned images, “special effects”, file format conversion, image enhancement through color reduction/promotion/dithering, and GUI (Graphical User Interface) functions. The toolkit supports the following bit depths internally: 1, 4, 8-bit gray level, 8i, 9-16-bit gray level, 24, and 32. The toolkit can also read all the additional bit depths possible for the supported file formats.

With ImageGear , you are assured of the high quality and performance you expect from AccuSoft, the industry leader in image processing toolkits. In addition, ImageGear’s integrity is backed by the AccuSoft Image Guarantee: ImageGear is guaranteed to read all valid raster images in the supported formats. See the AccuSoft Image Guarantee for more information.

In This Guide

This manual consists of two Volumes, organized as follows:

Volume 1: User Guide provides the following information about ImageGear:

- The Introduction chapter includes information about ImageGear features, improvements, ImageGear components structure, plus information about AccuSoft’s Product Support Services and descriptions of AccuSoft’s other products.
- The Installation chapter explains how to install ImageGear. It also describes the structure of installed ImageGear files and the ImageGear Licensing process.
- The Getting Started chapter demonstrates how to incorporate ImageGear’s imaging capabilities into your own applications by using the sample source imaging programs provided with your toolkit.
- The Understanding Bitmap Images chapter helps application developers to thoroughly understand Device-Independent Bitmaps (DIBs), the memory structures that are at the heart of ImageGear’s architecture. The majority of ImageGear functions operate either directly or indirectly upon DIBs. Device-Dependent Bitmaps (DDBs), the meaning of pixels (the building blocks of bitmaps), and all color and gray-scale representations supported by ImageGear are also discussed.
- The Using ImageGear chapter explains how to call an ImageGear function, including setting up the variables and structures passed as arguments, and details which functions to use to perform the variety of imaging tasks

mentioned in the Overview above.

- The Chapter Creating Your Imaging Application guides you through the steps of creating, testing, and finalizing your application. This discussion includes declaring compile/link switches and default directories, activating debug mode, and interpreting the function success/error return codes.
- File Format Reference chapter provides detailed information on each of the file formats supported by ImageGear, its control parameters, supported features and compressions, plus “at-a-glance” tables that describe the features supported by each format and provide guidelines on which formats may be most suitable when speed, storage space, or similar considerations are important.
- The AccuSoft Image Guarantee chapter explains the ImageGear Image Guarantee and how to identify the imaging problem if it occurs.

This volume also includes the following appendices:

- Appendix A: Function Error Return Codes lists the error codes returned by ImageGear functions. These error codes are classified in different groups and numbered in alphabetical and numerical orders.
- Appendix B: Non-Image Data Storage provides detailed info about TIFF, JPEG, EXIF-TIFF, EXIF-JPEG, IPTC, PNG and GIF formats non-image data storage.
- Software License Agreement provides the AccuSoft’s Software License Agreement and Software Support and Maintenance Policy documents.
- Glossary
- Bibliography

Volume 2: API Reference provides detailed information about each function or control parameter of the following ImageGear Components:

- Core Component API Function Reference
- ART Component API Function Reference
- MD Component API Function Reference
- PDF Component API Function Reference

This manual also provides an Index. The contents of this manual are also included in PDF Format on the Installation CD. The PDF files for ImageGear Unix and the components are copied to the `$HOME/AccuSoft/ImageGear15/Pro/Documentation/Help/` directory when you install the product.

See also the sections Description of Installed Files and In This Guide.

What’s New in ImageGear

ImageGear 15.3 for Windows 64-bit introduces the ARTX and HD Photo components, among other significant enhancements. This section provides an overview of the new features added into ImageGear v15.3.

What's New in ImageGear 15.3

ImageGear v15.3 introduces the following new features and enhancements:

- Added the ability to delete affected marks after burning them in.
  - Art 2.0 flag ART_BURN_IN_DELETE_MARKS added for function **ART_image_burn_in**
- Added the ability to use interpolation with rotation (**IG_IP_rotate_any_angle_ex**)
- Added the ability to count the number of unique colors in the specified rectangular area of the image with the addition of **IG_IP_color_count_get**
- Added PDF Layer Capability. In addition to the following API, please also see the **Working with PDF Layers** section in the Using ImageGear chapter.
  - **IG_PDE_content_create,**
  - **IG_PDE_element_get_dictionary**,
  - **IG_PDE_element_get_unique_id**,
  - **IG_PDF_dictionary_create**,
  - **IG_PDF_dictionary_get_layer**,
  - **IG_PDF_dictionary_get_layer_count**,
  - **IG_PDF_dictionary_get_unique_id**,
  - **IG_PDF_dictionary_get_vis_policy**,
  - **IG_PDF_dictionary_release**,
  - **IG_PDF_layer_create**,
  - **IG_PDF_layer_get_unique_id**,
  - **IG_PDF_layer_release**,
  - **IG_PDF_layer_remove**
- Added the ability to automatically stitch all image tiles during loading for the following formats (please also see Automatic Tile Stitching in the Using ImageGear chapter):
  - **IBM AFP**
  - **IBM IOCA**
  - **IBM MO:DCA**
  - **JPEG2000**
  - **JPX**
  - **Motion JPEG2000**
  - **TIFF**

- Added support for DICOM overlay data. Use **MED_overlay_apply**() function to load graphic overlays from DICOM DataSet to ART annotations. Use **MED_overlay_extract**() function to save ART annotations into DICOM DataSet as overlay data. Also see Working with Overlays.
- Added the new DICOM control parameters: DETECT_CONTINUOUS_RLE, LOAD_SAVE_PIXDATA_TAG
- Added the ability to update a DICOM DataSet without updating Pixel Data (**MED_DCM_DS_update_file**). Also see Updating DICOM DataSet without updating Pixel Data.
- A new "File Formats" user guide chapter that provides comprehensive, useful information on the file formats supported by ImageGear.
- New PDF component functionality to allow access to the font style and metrics. The following was added:
  - Two new functions were added to the **HIG_PDF_STYLE** object:
    - **IG_PDF_style_get_color**
    - **IG_PDF_style_get_font**
  - The new object, **HIG_PDF_FONT**, was added as a handle to a PDF font object that is used to draw text on a page.
  - The enumeration **enumIGPDFCharset** was added.
  - Two new structures, **AT_PDF_FONT_STYLES** and **AT_PDF_FONT_METRICS**, were also added.
- PDF word searching was greatly improved to support searching for substrings. The following were added to the **HIG_PDF_PAGE** object:
  - **IG_PDF_page_get_rotation**
  - **IG_PDF_page_get_crop_box**
- Updated the PDF libraries to be able to rasterize password protected PDFs so that the user can print with non-PDF-native printing.
- Added new display control parameters to specify the size of offscreen drawing surface for **IG_display_option_get**/**IG_display_option_set**:
  - IG_DISPLAY_OPTION_OFFSCREEN_WIDTH
  - IG_DISPLAY_OPTION_OFFSCREEN_HEIGHT.
- Improved samples/UI behavior to offer enabling ICC profiles when needed:
  - **IG_cpm_image_embedded_profile_check**
  - **IG_gui_colormismatch_warning**
  - **IG_gui_get_showiccwarning**
  - **IG_gui_switch_showiccwarning**
  - **IG_gui_get_enableicc**

- **IG_gui_switch_enableicc**

- Added support for LZW compressed RAW images.
- Greatly improved the performance of **IG_IP_thumbnail_create**
- **An improved API for working with grayscale Look Up Tables (LUTs).** It can be used for displaying a specific range of image intensities, as well as for tone correction. Please see the sections Working with Grayscale Look Up Tables and Grayscale Look Up Tables as well as Core Component API Function Reference chapter for `IG_LUT_...`, `IG_dspl_LUT...` and `IG_image_LUT_...` functions reference.
- **Support for non-linear DICOM Modality and Value Of Interest Look Up Tables (LUTs).** The new API can be used for creating grayscale LUTs according to DICOM display attributes, such as VOI LUT, Modality LUT, Presentation LUT, etc. Please see the section Medical Component Grayscale Look Up Tables and MD Component API Function Reference chapter for `MED_DCM_DS_LUT_...`, `MED_VOI_window_...`, `MED_display_grayscale_LUT_...`, `MED_PS_GSDF_LUT_...` and **`MED_modality_transform_apply()`** functions reference.
- **Improved support for Camera Raw images.** The color reconstruction engine has been improved. All reconstruction codes are now incorporated in the toolkit, so no additional processing is required on the application side. A number of new camera raw formats have been added. Please see Working with ImageGear Digital Camera RAW Filter section, Adobe DNG and TIFF sections of File Format Reference Chapter, as well as the section ImageGear Supported Digital Camera RAW Formats.
- **Support for both DICOM raster and vector (waveform) images**. ImageGear loads waveform images in vector form. You can use the Vector component API to modify waveform images. ImageGear reads and displays waveform data (graphs), as well as axes and some annotations.

ImageGear Overview

ImageGear and its Components provide the most comprehensive set of imaging features available in the industry. It provides a complete set of over 400 functions focused on the following features:

- High Speed Display
- Image Loading and Saving
- Printing
- Image Processing
- Special Effects
- Pixel Access
- ROI Processing
- Format Conversion
- Compression

- Transitions
- Native CMYK Support
- Database
- Built-in Internet Support (URL, FTP, Gopher)

High Speed Display

ImageGear gives you complete control over how your application displays each of its images. Among the attributes you can set on an image-by-image basis are:

- Auto-color reduction for low color modes
- High-quality display for all video modes
- Contrast/Brightness control
- Display effects (wipes, blocks, etc.)
- Color reduction (several types)
- Transparency
- Gamma correction
- Dithering
- Rotation
- The portion of the image to display (this array of pixels within the DIB bitmap is called the Image Rectangle)
- How to fit the Image Rectangle to the Device Rectangle
- The background fill pattern and color to use (for any area of the Device Rectangle left vacant by the image)
- Region within the display area the image is to be displayed (this area is called the Device Rectangle)
- Center image in window
- Fit to width, height, window
- Precision scrolling
- Sub-region display
- Auto-aspect ratio
- Preserve black and preserve white display for 1-bit images
- Sub-second display rotation supported
- Faster image display
- Anti-alias display
- Huge image display capability
- Merge 2 Images during display
- 4x faster scale-to-gray

- Also, by using the image's LUTs (Look Up Tables), you can cause the colors in the image's color palette to be translated to other colors you select (and therefore you can, for example, display an 8-bit gray-scale image in any 256 colors you choose).
- All of the above display attributes affect the display only. They do not alter either the image bitmap or the color palette in the DIB.
- In addition, any time your image is being displayed, you can center, zoom, or scroll it from within your application.

See Also:

Loading Images

Saving Images

Displaying Images

Thread Safety

Understanding Display Options

Color Reduction

Color Promotion

Contrast Alteration

Working with Format Filters

Working with ImageGear Color Profile Manager

Working with Global Control Parameters

Working with Color Space Conversion Engine

Working with Grayscale Look Up Tables

Working with Multi-page Images

Working with Stripped Images

Working with Tiled Images

Working with Run Ends Format

Working with ImageGear Digital Camera RAW Filter

Image Loading and Saving

ImageGear now supports over 200 raster file formats. This far exceeds all other image toolkit vendors. AccuSoft also is the only company willing to guarantee that all images in the listed formats will be displayed correctly!

See Also:

Loading Images

Saving Images

Displaying Images

Printing

- Print to any Windows printer with complete control.
- Auto-color reduction for high-quality printing.
- Single- or multi-page printing.
- ImageGear automatically overcomes several inherent limitations of printer drivers to ensure your images print at the highest quality possible.
- ImageGear's printing functionality includes automatic sizing to full page, half, quarter, eighth, and sixteenth, with auto page centering, or specific placement.
- Printing many images to a single page is as simple as specifying the location of each image.
- Images can be printed to any location and at any size desired.
- ImageGear includes a sample to illustrate printing a single image onto multiple pages for creating posters of any size.

See Also:

Printing Images

Image Processing

- Region of interest (ROI) support for basic rectangles as well as ellipses, polygons, and other shapes.
- The most complete color space support, including color space conversions, color separation and combination, support for any color space found in any of the 200 supported formats, and support for color spaces such as CMYK without conversion to RGB for perfect fidelity.
- Powerful color reduction for maximum quality and minimum size.
- Encryption and decryption for entire image or any part of image.
- Matrix convolutions of any size with pre-defined or user-supplied matrix values.
- Special effects, automatic image correction, intelligent re-sizing, and much more.

See Also:

Image Processing

Core Component API Function Reference

Special Effects

ImageGear provides the following special effects:

- Blur
- Emboss
- Chroma Key
- Add Noise
- Texture Tile
- Diffuse
- Motion
- Spotlight
- Stitch
- Watermark
- Pixelate
- Posterize
- Twist

See Also:

Special Effects

Pixel Access

ImageGear is equipped with several functions that will allow you to get and set the values of individual pixels, rows or columns of pixels, and rectangular groups of pixels. This family of functions is referred to as the “pixel access” functions. For each kind of pixel access, you can obtain the value(s) of a pixel or group of pixels, or set the value(s) of a pixel or group of pixels.

See Also:

Pixel Access Operations

ROI Processing

- Specify rectangular ROI for nearly all image processing functions.
- Specify arbitrary ROI for most image processing functions. Functions included to create certain shape types such as ellipse, polygon, and freehand.
- Create 1 bit mask image for identifying which pixels to include/disclude from image processing algorithms.

See Also:

Image Processing

Core Component API Function Reference

Format Conversion

ImageGear now supports over 200 raster file formats, by far exceeding all other image toolkit vendors.

You may want to convert a file from one format to another. You can do this by saving the original file to its desired format by setting the `nFormatType` parameter to the appropriate value in the saving function. For more information on converting issues, see the section Saving Images.

AccuSoft is also the only company willing to **guarantee** that all images in the listed formats will be displayed correctly.

See Also:

File Format Reference

ImageGear Supported File Formats Reference

ImageGear Supported Bit Depths

Compression

ImageGear supports most of the industry-standard compression algorithms.

See Also:

ImageGear Supported Compressions Reference

ImageGear Supported File Formats Reference

Transitions

ImageGear has the most advanced and flexible image-display technology available. It is also the fastest, with much of the functionality coded in assembler for optimal speed. One component of this is the ability to transition from one image to another in a more interesting fashion than simple replacement. Sometimes referred to as “transitions” or “wipes”, these features are useful for slide shows, or any type of product where the image itself is the focus. For this type of product, ImageGear provides a set of functions for migrating from one image to another in an interesting way.

The transition support in ImageGear includes 29 types of transitions with control over the “granularity” and speed. Granularity refers to the size of the object used or the smoothness of the transition. The speed is the total time used to make the transition from one image to another. Of course, all other display parameters are also available for controlling the transitions.

See Also:

Image Transformation

Image Analysis

Special Effects

Blending and Combining Images

Image Correction

Image Maintenance

Native CMYK Support

- Images stored in CMYK format can be loaded into ImageGear without being converted to RGB. This ensures the original color information is maintained with perfect fidelity.
- Images can be saved to formats that support CMYK color space.
- Images stored in CMYK can be optionally converted to RGB when loaded (default for STD), or converted to RGB at any time after loading.
- Basic image processing, image access, display, and other functions work with native CMYK image data.
- CMYK images can be color separated or converted to several other color spaces.

See Also:

Working with Color Space Conversion Engine

Database

- Load images from memory in 200 plus formats.
- Load images from file with any offset for images embedded in a database.
- Decode images using various compression algorithms without specific format headers.
- Import/export images from/to various types of memory formats.
- Save images to files at specified offsets.

Annotations

The ImageGear ART component integrates with ImageGear and supports the Wang annotation specification for storing annotations in TIFF. However, this is not a requirement; annotation data may be stored anywhere. The simple API makes adding and managing annotations easier than ever. ART works as an overlay to loaded images. ART annotation marks can include vector graphics, text, bitmaps, user-defined marks and others. There are many built-in features such as automatic handling of object creation, a toolbar that you can customize, attribute dialog boxes for each type of mark, built-in internationalization and more. To use ImageGear ART functionality you should attach ImageGear ART component. Please see ImageGear ART Component.

Built-in Internet Support (URL, FTP, Gopher)

AccuSoft's ImageGear product has Internet Support for Web pages. You can take advantage of AccuSoft's new Internet support to enable your product on the Web.

ImageGear Supported File Formats

ImageGear supports a great number of the most popular graphics file formats with different compressions, preventing compatibility problems before they arise. These ImageGear-supported file formats are described in detail in the File Format Reference Chapter. See also the section ImageGear Supported File Formats Reference.

The complete list of supported formats (as of this writing) is below:

Format	Read/write?	Description
Adobe DNG	Read	Adobe Digital Negative format
Adobe PDF	Read-write	Adobe Portable Document Format, with ImageGear PDF Component only.
Adobe PS	Read-write	Adobe PostScript language files level III. Requires ImageGear PDF Component.
Adobe PSD	Read-write	Adobe Photoshop
AFX	Read	AutoFX format
BMP	Read-write	Standard Windows format including OS/2 bitmap format.
BTR	Read-write	Brooktrout FAX format, Group III or Group III 2D or 301
CAL	Read-write	CALS raster, Type I
CLP	Read-write	Windows Clipboard format for raster images
CUT	Read	Dr. Halo CUT format
DCRAW	Read	Raw images created by many digital cameras. See the sections Working with ImageGear Digital Camera RAW Filter and ImageGear Supported Digital Camera RAW Formats.
DCX	Read-write	Intel multi-page FAX format
EPS	Read-write	IG without a PDF component allows for the reading of the preview portion of EPS files and writes EPS files. ImageGear PDF component allows for the reading of any EPS files.
EXIF-JPEG	Read-write	Exchangeable image file format for Digital Still cameras. ImageGear also allows you to process non-image data that is usually encoded in this file format. See the section Non-Image Data Processing.
EXIF-TIFF	Read-write	Uncompressed EXIF is recorded in TIFF Rev. 6.0 formats with two pages - the first is the main image, the second is a thumbnail (if it present).
GEM	Read	GEM Paint Raster format

Format	Read/write?	Description
GIF	Read-write	ImageGear supports every aspect of GIF including animated and transparent. Requires ImageGear LZW Component.
Group 3	Read-write	Proprietary formats with custom headers followed by Group III data.
Group 3 2D	Read-write	Proprietary formats with custom headers followed by Group III 2D data.
Group 4	Read-write	Proprietary formats with custom headers followed by Group IV data.
IBM AFP	read	Meta file format that may include raster and/or text data. This type format belongs to IBM's format family, which also includes IBM IOCA, IBM MO:DCA, and IBM MMR compression.
IBM IOCA	Read-write	Image Object Content Architecture, CCITT Group 3, CCITT Group 3 2D, CCITT Group 4, IBM MMR, uncompressed, ABIC (with only).
IBM MO:DCA	Read-write	Mixed Object Document Content Architecture format, CCITT Group 3, CCITT Group 3 2D, CCITT Group 4, IBM MMR, uncompressed, ABIC
ICO	Read-write	Windows ICON format
IFF	Read-write	Electronic Arts format
IMG	Read	Xerox raster format, Group IV
IMR	Read	IMRS-Raster image
IMT	Read-write	IMNET format, Groups III and IV.
JPEG	Read-write	ANSI-standard, full-color image format. IG supports all aspects of this format including lossless and progressive. JFIF - industry standard header for JPEG compressed images.
KFX	Read	Kofax format, Group IV
LV	Read	LaserView, LaserData, Group IV
MSP	Read	Microsoft Paint format
NCR	Read-write	
PBM	Read-write	Portable BitMap (Unix graphic file format)
PCD	Read-write	Kodak format for high resolution photographs
PCX	Read-write	PC Paintbrush format
PGM	Read-write	Portable BitMap (Unix graphic file format)

Format	Read/write?	Description
PNG	Read-write	Portable Network Graphics, intended as successor to GIF
PNM	Read-write	Portable BitMap (Unix graphic file format)
PPM	Read-write	Portable BitMap (Unix graphic file format)
RAS	Read-write	SUN raster format
RAW	Read	Raw pixel data storage image formats, proprietary image formats
SGI	Read-write	Silicon Graphics raster format
TGA	Read-write	Targa file format. ImageGear supports many aspects of this format including alpha channel.
TIFF	Read-write	Tagged Image File Format. IG supports every aspect of this format.
TXT (ASCII Text)	Read	Converted to raster image. Control font, point size, page break, margins, attributes etc.
WBMP	Read-write	Optimized to support mobile computing devices that use the Wireless Application Protocol (WAP).
WMF	Read-write	Windows standard metafile format. IG converts non-raster data to raster during load.
WPG	Read	Wordperfect Graphic Format. IG reads bitmaps stored in this format
XBM	Read-write	X Windows Bitmap format
XPM	Read-write	X Windows Pixmap format
XWD	Read-write	X Windows Dump format

NOTE: AccuSoft is always willing to consider supporting a new format. If you are interested in discussing support for your own format, please call the AccuSoft Sales Department at (508) 351-9092.

ImageGear Support for DCRAW Formats

ImageGear supports 100+ Digital Camera RAW formats.

ImageGear supports reading raw images created by many digital cameras. Support for raw images includes reading of image pixel data, color reconstruction of raw images, and reading of metadata stored with the images.

Since there are many flavors of raw formats, the list is continuously growing as new vendors/models enter the market, and the formats share many similarities, ImageGear treats all DCRAW formats as one format: `IG_FORMAT_DCRAW`. Details regarding manufacturer, model, etc, are available as metadata.

Please see the section Working with ImageGear Digital Camera RAW Filter to know how to work with DCRAW formats using ImageGear functionality.

For the list of supported DCRAW formats and the level of support please see the section ImageGear Supported Digital Camera RAW Formats.

ImageGear Components

ImageGear functionality is broken into components. This provides extra flexibility in packaging and licensing ImageGear technology. Each component encapsulates particular functionality of the product. All components are enabled when you evaluate or purchase the product (there is an expiration date for evaluation version), but different licensing fees may be associated with different components for deployment of your application. Please contact AccuSoft's Sales for more information about deployment licensing fees.

You need to attach a component (other than the core component) before using it. See the Samples for a demonstration of attaching the components. The sample code has been provided so that you may simply cut and paste the attaching component code into your own application.

Please, see ImageGear Licensing for full information about ImageGear and its component licensing process. See also the Software License Agreement.

The API used to attach components, **`IG_comm_comp_attach()`** and **`IG_comm_comp_check()`**, are described in Core Component API Function Reference and in the section Using ImageGear Component Manager.

NOTE: In your ImageGear application code, you will need to use the short component names. Please see the section ImageGear Component Names for the full list.

ImageGear Component Descriptions

The following sections summarize the ImageGear components that are currently available. For more information on these components, see the documentation provided with each one.

- ImageGear ART Component
- ImageGear Medical Component
- ImageGear PDF Component

NOTE: You can only attach one ART component. ARTX cannot be attached when ART is already attached.

ImageGear ART Component

The ImageGear ART component includes low-level functions for the fine-tune image annotating and high-level functions for the most typical operations. ART creates annotations on a “layer” that sits “above” the image or document that is to be annotated. The original image and the annotations can be stored separately or merged together permanently.

NOTE: You can only attach one ART component. ART cannot be attached when ARTX is already attached.

For detailed information about using the ART component, see the section Using ImageGear ART Component and the chapter entitled ART Component API Function Reference.

ImageGear LZW Component

This component allows you to work with LZW compression images.

See Also:

GIF Non-image Data Structure

This component allows you to load and save HD Photo images.

See Also:

HD Photo

ImageGear Medical Component

The ImageGear Medical (MD) component is a full-featured ImageGear component. It supports the DICOM format, contains a custom API, and includes expanded image processing capabilities beyond those of the baseline ImageGear library.

The format support of the MD component includes loading and saving monochrome, palletized, and true color medical images using the following file formats:

- DICOM 3.0 Part 10-compliant images
- DICOM 3.0 Raw Format (non-Part 10-compliant)

In addition, your application will continue to support all ImageGear-supported file formats, allowingt you to convert an image of a different format to a medical image format, and vice- versa.

NOTE: The ImageGear MD component is available for Linux platforms.

The ImageGear MD component is backed by the AccuSoft Image Guarantee. The component is guaranteed to read all valid raster images in the supported formats.

See Also:

Using ImageGear MD Component

MD Component API Function Reference

ImageGear PDF Component

The ImageGear PDF Component allows you to load, save and process **PDF (Portable Document Format)** file format images using core ImageGear and other IG Components functionality.

This Component also allows you to read vector data from **PS (PostScript)** and **EPS (Encapsulated PostScript)** file formats (core ImageGear Component allows you to read only raster data from these two formats).

In addition, this PDF Component provides the ability to extract text from loaded PDF and PS image files.

NOTE: The Adobe®-based PDF Component is applicable for Linux (i386), Solaris 32-bit, and AIX Unix Platforms.

Linux for PowerPC, HP-UX 32-bit & 64-bit, HP-UX for Itanium, and Sun Solaris 64-bit use the GS PDF Component .

See Also:

Using ImageGear PDF Component

PDF Component API Function Reference

ImageGear Samples

Your ImageGear toolkit also contains a directory with sample imaging application programs and images. You may copy and modify them as you wish. You can also use them as templates for developing your own applications. The sample images may also be used for any purpose, such as testing your applications as you develop them.

Answering Your ImageGear Questions

After reading the manual, one of the best ways to begin to answer your questions is by reading the release notes, which can be accessed through the ImageGear application menu.

See Also:

AccuSoft Customer Support

AccuSoft Customer Support

AccuSoft includes one year of the support and subscription program with the ImageGear product at no additional cost. If you desire a higher level of support, we offer additional Support programs designed to meet the diverse requirements of AccuSoft customers. Please visit AccuSoft's Web site at

http://www.accusoft.com/support/s_about.asp

or call our Sales Department at (508) 351-9092.

You can also e-mail questions to **sales@accusoft.com** .

See also the section Software Support and Maintenance Policy in "Software License Agreement" to get information about AccuSoft's customer support policy.

Before Contacting Customer Support

- Double check the **Manuals** provided with this product or check the on-line documentation.
- Read the release notes, which can be accessed through the ImageGear application menu.
- Refer to the following chapters: Getting Started, Using ImageGear, and AccuSoft Image Guarantee. These chapters contain a great deal of information on both programming your application and identifying image problems.
- Take a look at the sample programs included with your product.
- If you still need technical support assistance, please follow the instructions provided in the AccuSoft Customer Support section.

Internet Services

Visit the AccuSoft website (**www.accusoft.com**) for useful information, including the following areas:

- Product Information
- Product Downloads
- Customer Support
- Demos and Evals
- On-line Documentation

Installation

This Chapter provides you with all necessary information about how to install/ uninstall ImageGear in the following sections:

- Minimum Requirements
- Installing ImageGear from the Web Site
- Directory Structure
- Description of Installed Files
- Uninstalling ImageGear
- ImageGear Licensing

Minimum Requirements

Vista and Visual Studio 2005

Microsoft recommends running Visual Studio 2005 with administrator privileges under Vista. To properly use the ImageGear Samples please run Visual Studio 2005 with elevated permissions.

To run Visual Studio 2005 with elevated administrator permissions first confirm you are a member of the "Administrators" group on the local machine.

Then right-click the Visual Studio 2005 shortcut and select the "Run as administrator" option from the context menu. Click ‘Continue’ to the UAC prompt.

To ensure you have the latest updates, after running Visual Studio 2005 choose ‘Check for Updates’ from the Help Menu.

Before installing ImageGear for Unix, make sure that your computer system meets the following minimum requirements.

AIX

- POWER, POWER2, or PowerPC platform running AIX 5.3 operating system
- X11R6 include files & libraries
- IBM xlC compiler v7.0
- Motif 1.2 optional (to compile the samples)

HP-UX

- HP 9000/785 series platform running HP-UX 11.x operating system
- X11R6 include files & libraries
- HP aC++/ANSI C HP-UX Version 11 compiler
- Motif 1.2 optional (to compile the samples)

HP-UX Itanium 64 (32-bit or 64-bit)

- HP IA64 series platform running HP-UX 11.x operating system
- X11R6 include files & libraries
- HP aC++/ANSI C HP-UX Version 11
- Motif 1.2 optional (to compile the samples)

Linux (i386)

- Intel based platform running one of the following distributions:
  - Red Hat Enterprise Linux 4
  - Ubuntu 6.06
- X11R6 include files & libraries (openmotif-devel package under RHEL) required to build GUI samples
- Detailed requirements:
  - 2.4 or 2.6 kernel
  - Following libraries built with thread local storage enabled:
    - libstdc++.so.5
    - libX11.so.6
    - libc.so.6
    - libm.so.6
    - libgcc_s.so.1
    - libXau.so.6
- GCC 3.3 compiler
- Motif 1.2 optional (to compile the samples)

Linux (PowerPC)

- PowerPC 64 based platform running Red Hat Enterprise 3
- X11R6 include files & libraries
- GCC 3.2 compiler
- Motif 1.2 optional (to compile the samples)

Solaris (32-bit or 64-bit)

- Sun SPARC platform running Solaris 5.9 operating system
- X11R6 include files & libraries
- SUN Work Shop Pro C/C++ 5.6
- Motif 1.2 optional (to compile the samples)

Installing ImageGear from the Web Site

To install ImageGear, download an electronic version from our Web Site www.accusoft.com. Please contact AccuSoft Customer Service at (508) 351-6956 for instructions on downloading your specific version of ImageGear for Unix.

The name of the ImageGear installation package for Unix is one of the following (the first for 32-bit platforms, and the second for 64-bit platforms):

ImageGear15.3.tar.gz
ImageGear15.3-64.tar.gz

To install ImageGear, perform the following steps:

1. Move the installation file into your home directory **$HOME** and run: **tar zxvf {installation filename}**

2. In order for ImageGear to work, there must be a license file installed. The license key is kept in the file with the name **accusoft.<solution name>.imagegear**. ImageGear installation comes with predefined solution name as **'AccuSoft 1-101-15'** for 32-bit, or **'AccuSoft 1-105-15'** for 64-bit. The blank **accusoft.AccuSoft 1-101-15.imagegear** license file is included in the installation. The license obtained from the AccuSoft web site needs to be inserted in this file. When it's done the variable **IMAGE_GEAR_LICENSE_DIR** has to be defined and should contain the path to the license file (for example **$HOME/AccuSoft/ImageGear15/Pro/Unix/**). If the **IMAGE_GEAR_LICENSE_DIR** variable is not defined, ImageGear will look for the license file in **/usr/lib/** directory.

3. In order to let the linker find the shared libraries, define the **LD_LIBRARY_PATH** environment variable. For example:

   ```
   LD_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/
   Linux
   ```

4. The ImageGear application uses the **IMAGE_GEAR_LIBRARY_PATH** variable to set a path to the ImageGear libraries. For example:

   ```
   IMAGE_GEAR_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/
   Bin/Linux
   ```

Automated Installation Script

1. Extract the **ImageGear15.3.tar.gz** for 32-bit or **ImageGear15.3-64.tar.gz** for 64-bit, by running one of the following:

   ```
   tar zxvf ImageGear15.3.tar.gz
   tar zxvf ImageGear15.3-64.tar.gz
   ```

2. Located in the **AccuSoft/ImageGear15/Pro/** folder (**AccuSoft/ImageGear15-64/Pro/** folder for 64-bit platforms) after extracting is the **install.sh** script, running it:

   ```
   HPUX (C Shell)-> ./install.sh
   Solaris (Korn Shell) -> .install.sh
   ```

will search for and modify the current user's shell profile files to add ImageGear's enviroment variables for the current folder and attempt to run the java AccuSoft License Manager. Optionally a destination directory can be used as an argument. For example:

```
. install.sh $HOME/ImageGear15/Pro
```

In this case the contents of the **AccuSoft/ImageGear15/Pro/** directory will be copied to the specified directory and the enviroment variables and licensing is done for this location.

Manual Installation

1. After downloading an evaluation copy of ImageGear for Unix, move the installation file into your home directory **`$HOME`** and extract the files. A typical method is to run:
   `tar zxvf {installation filename}`
2. In order for ImageGear to work there must be a license key installed. This license key is kept in the file with the name: **`'accusoft.<solution name>.imagegear'`**. ImageGear for Unix comes with a predefined solution name: **`'AccuSoft 1-101-15'`** `or` **`'AccuSoft 1-105-15'`** `for 64-bit`. The blank **`"accusoft.AccuSoft 1-101-15.imagegear"`** license file is included in the installation. A license obtained from the AccuSoft web site needs to be inserted in this file. To do this:
   a. Run the EnvParamsUtility included in the installation. This is located at: **`$HOME/AccuSoft/ImageGear15/Pro/Utils/<Unix OS>/EnvParamsUtility`**. To run it go to the directory above and type: **`./EnvParamsUtility`**. This will generate a **Hardware ID** of the form: **`mac-00-A0-C9-73-06-20`**. You will use this **Hardware ID** to generate a **License Key** in the next step.
   b. For evaluation start the **WebEvalClient** Utility on AccuSoft's website at: **`https://www.accusoft.com/licensing/v1/webevalclient/product.aspx.`** If you have already purchased development license, start the **WebEvalClient** Utility on AccuSoft's website at: **`https://www.accusoft.com/licensing/v1/WebProductClient/login.aspx.`** Select the **UNIX OS** type (or Java to obtain IGJava license), and Login with the email and password you registered with AccuSoft. Enter the **Hardware ID** from the previous step (not required for Java), and you will be given the **License Key**. You will need to put this **License Key** in the license file in the next step.
   c. Open the **`'accusoft.AccuSoft 1-101-15.imagegear'`** file in an editor. It's located at: **`$HOME/AccuSoft/ImageGear15/Pro/Unix/accusoft.AccuSoft 1-101-15.imagegear`**. Copy the License Key from the previous step and save the file.
3. Now that the license file is created the variable **`IMAGE_GEAR_LICENSE_DIR`** has to be defined and should contain the path to the license file (for example: **`$HOME/AccuSoft/ImageGear15/Pro/Unix/`** assuming the default install). If

the **IMAGE_GEAR_LICENSE_DIR** variable is not defined ImageGear will look for the license file in **/usr/lib/** directory. In order to let the linker find the shared libraries, you need to define:

- For Linux and Solaris, LD_LIBRARY_PATH environment variable. For example: **LD_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/Solaris**
- For AIX: LIBPATH environment variable. For example: **LIBPATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/AIX**
- For HP-UX: SHLIB_PATH environment variable. For example: **SHLIB_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/HP-UX**

The ImageGear application uses the IMAGE_GEAR_LIBRARY_PATH variable to set a path to the ImageGear libraries. For example: **IMAGE_GEAR_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/Solaris**

Here is an example Linux .bash_profile with all the necessary variables defined:

```
# User specific environment and startup programs
PATH=$PATH:$HOME/bin:.
BASH_ENV=$HOME/.bashrc
USERNAME=""
LD_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/
Linux
IMAGE_GEAR_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/
Bin/Linux
IMAGE_GEAR_LICENSE_DIR=$HOME/AccuSoft/ImageGear15/Pro/Unix

export USERNAME BASH_ENV PATH LD_LIBRARY_PATH
IMAGE_GEAR_LIBRARY_PATH
IMAGE_GEAR_LICENSE_DIR
```

Installing the Adobe PDF Component

The Adobe PDF component is available for Linux, Solaris, and AIX 32-bit platforms. For 64-bit platforms or for HP-UX 32- or 64-bit platforms, see the next section, Installing GS PDF Component.

It is necessary to install additional files for the optional PDF extension. Because the ImageGear for Unix PDF and PS support implementation is based on Adobe PDF and PostScript library, some additional steps are necessary for proper installation.

The following environment variables must specify a path to the PDF component files:

- For Linux and Solaris: LD_LIBRARY_PATH environment variable. For example:

  ```
  LD_LIBRARY_PATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/Linux
  ```

- For AIX: LIBPATH environment variable. For example:

```
LIBPATH=$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/AIX
```

See PDF Component API Function Reference for more ImageGear PDF initialization details.

Installing GS PDF Component

If you are using HP-UX Unix platform or any 64-bit platform, it is necessary to install additional files for the PDF component. Because the ImageGear for HP-UX Unix and 64-bit Unix PDF and PostScript support implementation is based on Artifex Ghostscript, some additional steps are necessary to be done for the proper installation.

There are two possible ways to get this done:

1. The subdirectory **/accugs** contains the files necessary to run the Unix version of Ghostscript. It is possible to set up the environment variable **`IMAGE_GEAR_ACCUGS`** to point to this directory. For example: **`IMAGE_GEAR_ACCUGS=$HOME/AccuSoft/ImageGear15-64/Pro/Unix/Bin/Solaris/accugs`**
2. If this variable is not defined, then the PDF component will search for the **/accugs** directory with all the files in the subdirectory **/usr/local**. So the alternate way is to copy **/accugs** to **/usr/local**.

Installing ImageGear JAVA

ImageGear Java package can be found in **`/AccuSoft/ImageGear15-64/Pro/Java/`**. To extract the ImageGear Java files, use a utility such as **`gzcat`**, **`zcat`** or **`gunzip`** and pipe the output to **`tar`**.

Example:

```
gunzip < java.tar.gz | tar xvf -
```

Linux users must use the GNU zip version of zcat.

Other Installation Directives

- There is a demo application included with this build located in: **`/Samples/filters`**. The "filters" sample requires Motif to be installed and provides basic dialogs to load and save images, load and unload extensions, change TIFF filter's control parameter, and so on. ImageGear itself does not require `Motif` but does require `libc`, `libm` and `libpthread`.
- You may load extensions not only with the shared version of ImageGear(**`libImageGear.so`**) but with the static version also (**`libImageGear.a`**).
- Please note that due to a conflict of the standard `BOOL` type used in a previous version of ImageGear with one of the X include files (`Xdm.h`), beginning with this release the `BOOL` type is not defined any more but `AT_BOOL` type is defined and should be used instead.

Extracting Files

To extract the Installation files, use a utility such as gzcat, zcat or gunzip and pipe the output to tar. To extract single files use:

```
/usr/bin/zcat < ImageGear*.tar.gz | tar xvf -
```

Directory Structure

The ImageGear for Unix installation will create the following directory structure, assuming that the home directory is the install directory:

NOTE: For 64-bit platforms, the directory stucture is `$HOME/AccuSoft/ImageGear15-64/...`

```
$HOME/AccuSoft/
$HOME/AccuSoft/ImageGear15/
$HOME/AccuSoft/ImageGear15/Pro

$HOME/AccuSoft/ImageGear15/Pro/Documentation/
$HOME/AccuSoft/ImageGear15/Pro/Documentation/Help/
$HOME/AccuSoft/ImageGear15/Pro/Documentation/Releasenotes/

$HOME/AccuSoft/ImageGear15/Pro/Java/
$HOME/AccuSoft/ImageGear15/Pro/Java/Bin/
$HOME/AccuSoft/ImageGear15/Pro/Java/Documentation/
$HOME/AccuSoft/ImageGear15/Pro/Java/Samples/

$HOME/AccuSoft/ImageGear15/Pro/Unix/
$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/
$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/AIX/
$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/HP-UX/
$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/Linux/
$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/Solaris/

$HOME/AccuSoft/ImageGear15/Pro/Unix/Include/
$HOME/AccuSoft/ImageGear15/Pro/Unix/Samples/

$HOME/AccuSoft/ImageGear15/Pro/Utils/
$HOME/AccuSoft/ImageGear15/Pro/Utils/AIX/
$HOME/AccuSoft/ImageGear15/Pro/Utils/HP-UX/
$HOME/AccuSoft/ImageGear15/Pro/Utils/Linux/
$HOME/AccuSoft/ImageGear15/Pro/Utils/Solaris/
$HOME/AccuSoft/ImageGear15/Pro/Utils/lib/
```

Description of Installed Files

All the files installed using the install procedure as described above will be

File Name	Description
accucb.h	C/C++ include file for ImageGear callback functions type
accucnst.h	C/C++ include file for ImageGear constants declarations
accufunc.h	C/C++ include file for ImageGear function prototypes
accuimg.h	C/C++ include file for ImageGear data types and macros
accuold.h	C/C++ include file for ImageGear backward compatibility declarations
accuosd.h	C/C++ include file for ImageGear operating system declarations
accustd.h	C/C++ include file for ImageGear data types and pointers
accuvers.h	C/C++ include file for ImageGear version declaration
artgui.h	C/C++ include file for the ImageGear ART component (ART GUI API)
gear.h	C/C++ include file for ImageGear core component
gearerrs.h	C/C++ include file for ImageGear error codes
gearestr.h	C/C++ include file for ImageGear error strings
geartags.h	C/C++ include file for ImageGear field tags used with tag callbacks
gif.h	C/C++ include file for ImageGear LZW component
i_ART.h	Public header file for the ImageGear ART Component
i_PDF.h	Public header file for the ImageGear PDF Component
i_PDFtp.h	C/C++ include file for ImageGear PDF Component
metadata.h	C/C++ include file for ImageGear metadata callback constants

installed in the directory structure shown above in the diagrams. The file names, file sizes, and usage requirements depend upon the version of the library you have purchased.

The following is a list of the files that are installed in **`$HOME/AccuSoft/ImageGear15/Pro/`**:

File Name	Description
`SoftwareLicense.rtf`	license agreement
`INSTALL.txt`	installation instructions

The following is a list of the files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Unix/**:

File Name	Description
accusoft.AccuSoft 1-101-15.imagegear	license file for 32-bit applications
accusoft.AccuSoft 1-105-15.imagegear	license file for 64-bit applications

The following is a list of the files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Unix/Bin/**:

File Name	Description
AIX/libImageGear.a	core static library for AIX
AIX/libImageGear.so -> libImgeGear.so.15	core shared library for AIX
AIX/libIGLZW.so -> libIGLZW.so.15	LZW shared library for AIX
AIX/libIGPDF.so -> libIGPDF.so.15	PDF shared library for AIX
AIX/libACE.so AIX/libAGM.so AIX/libBIB.so AIX/libBIBUtils.so AIX/libCoolType.so AIX/libJP2K.so AIX/libnserver.so AIX/libOPP.so AIX/libpdfl.so	PDF shared library for AIX
HP-UX/libImageGear.a	core static library for HP-UX
HP-UX/libImageGear.so -> libImgeGear.so.15	core shared library for HP-UX
HP-UX/libIGLZW.so -> libIGLZW.so.15	LZW shared library for HP-UX
HP-UX/libIGPDF.so -> libIGPDF.so.15	PDF shared library for HP-UX
HP-UX/accugs	PDF Ghostscript files for HP-UX

HP-UX/accugs/fonts	PDF Ghostscript font files for HP-UX
Linux/libImageGear.a	core static library for Linux
Linux/libImageGear.so -> libImgeGear.so.15	core shared library for Linux
Linux/libIGLZW.so -> libIGLZW.so.15	LZW shared library for Linux
Linux/libIGPDF.so -> libIGPDF.so.15	PDF shared library for Linux
Linux/libACE.so Linux/libAGM.so Linux/libBIB.so Linux/libBIBUtils.so Linux/libCoolType.so Linux/libJP2K.so Linux/libnserver.so Linux/libOPP.so Linux/libpdfl.so	PDF shared library for Linux
Solaris/libImageGear.a	core static library for Solaris
Solaris/libImageGear.so -> libImgeGear.so.15	core shared library for Solaris
Solaris/libIGLZW.so -> libIGLZW.so.15	LZW shared library for Solaris
Solaris/libIGPDF.so -> libIGPDF.so.15	PDF shared library for Solaris
Solaris/libACE.so Solaris/libAGM.so Solaris/libBIB.so Solaris/libBIBUtils.so Solaris/libCoolType.so Solaris/libJP2K.so Solaris/libnserver.so Solaris/libOPP.so Solaris/libpdfl.so	PDF shared library for Solaris

The following is a list of those files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Unix/Documentation/**:

File Name	Description

Help/IG_UNIX.pdf	electronic documentation (manual) for ImageGear
Help/LDK_Unix.pdf	electronic documentation (manual) for ImageGear licensing and deployment
Help/LDK_Java.pdf	electronic documentation (manual) for ImageGear licensing and deployment for Java
Help/AccuLicManager_UNIX_MAC.pdf	electronic documentation (manual) for AccuSoft License
ReleaseNotes	product release notes

The following is a list of those files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Unix/Include/**:

File Name	Description
accuimg.h	include files (header files)
accuosd.h	
accustd.h	
commapi.h	
cpmapi.h	
dspl.h	
fltr.h	
gear.h	
gearerrs.h	
gearestr.h	
geartags.h	
i_CORE.h	
i_LZW.h	
i_PDF.h	
metadata.h	
mpage.h	

The following table is a list of those files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Unix/Samples/filters/**. They are all the source files related to the sample applications that come with ImageGear:

dlg_dspl.c	dlg_dspl.h
dlg_error.c	dlg_error.h
dlg_extensions.c	dlg_extensions.h
dlg_formatlist.c	dlg_formatlist.h
dlg_ip.c	dlg_ip.h
dlg_jpeg.c	dlg_jpeg.h
dlg_open.c	dlg_open.h
dlg_pdf.c	dlg_pdf.h
dlg_saveas.c	dlg_saveas.h
dlg_tiff.c	dlg_tiff.h
filters.c	filters.h
Imakefile	

The following table is a list of those files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Unix/Samples/threads/:**

Imakefile	threads.c

The following table is a list of those files that are installed in **$HOME/AccuSoft/ImageGear15/Pro/Utils/**. They are all the source files related to the sample applications that come with ImageGear:

File Name	Description
<platform>/EnvParamsUtility	Environment parameters utility for <platform>
AccuSoftLicenseManager.jar	AccuSoft License Manager
lib	Java libraries used by ALM
<platform>/libLicHardwareNatJNI.so.15	Native AccuSoft License Manager library for <platform>

All the files installed using the install procedure as described above will be installed in the directory structure shown above in the diagrams. Depending on the version of the library you have purchased, the file names will change, as well

as, the file sizes and usage requirements. Also included are sample programs installed in the subfolder named **Samples**. These sample programs can be used as guides or examples of how to use the libraries in your applications. You may cut and paste freely from these sample programs into your own applications.

The following table is a list of IGJava files installed in **$HOME/AccuSoft/ImageGear15-64/Pro/Java/**:

File Name	Description
`Bin/igjava.jar`	IGJava Class Library
`Bin/igbean.jar`	IGJava Bean Library
`Documentation/LDK_Java.pdf`	Licensing help
`Documentation/Bean/manual.htm`	IGJava Bean help
`Documentation/CL/manualL.htm`	IGJava Class Library help
`Samples/accusoft`	IGJava samples' sources
`Samples/buildSamples`	Executable script to build all IGJava samples
`Samples/runBeanDemo`	Executable script to run BeanDemo
`Samples/runCLDemo`	Executable script to run CLDemo
`Samples/runThumbnailDemo`	Executable script to run ThumbnailDemo

NOTE: Java SDK is required to be installed in order for IGJava scripts to run. Java SDK can be downloaded from http://www.sun.com.

Sample Applications

In addition, a **SAMPLES** directory will be installed, if you have selected it during installation. Under this directory is a separate subdirectory for each sample. Each sample is a complete, executable mini application. In their totality, these applications offer you more breadth than most tutorials.

Uninstalling ImageGear

To uninstall the ImageGear Unix library, simply delete all the ImageGear files. You may issue the following command, if the files were installed in the default directory:

rm -rf Accusoft

If the automated install script was used, or if you manually added the IMAGE_GEAR environment variables to your shell profile, you can remove them now.

ImageGear Licensing

After the ImageGear installation is finished you need to obtain the **license key**.

If you are evaluating the product, use the AccuSoft ImageGear evaluation page:

https://www.accusoft.com/licensing/v1/WebEvalClient/product.aspx

Please log in using the same e-mail and password you have registered with on the AccuSoft Web site. After you supply the login credentials, you will be prompted to enter your **hardware ID**. Use the **EnvParamsUtility** utility from your ImageGear installation to obtain your hardware ID. Then click the **Generate** button and your **license key** will be automatically generated and displayed in the browser window. You can copy and paste it into your application.

NOTE: The Evaluation license enables you to use any features of the product on your machine for a limited period of time.

To extend the Evaluation license:

1. Login to the AccuSoft ImageGear evaluation page.
2. Click the **Submit extension request** button.

Your request will be considered and you will receive the e-mail notification when your license is extended. Then you would need to log in to the AccuSoft ImageGear evaluation page again to generate a new **license key**.

If you bought the **development** copy of the ImageGear product, use the AccuSoft license page for developers:

https://www.accusoft.com/licensing/v1/WebProductClient/login.aspx

You will be prompted to log in using the e-mail and password you have registered with on the AccuSoft Web site. Upon the login you will be presented with the page that has the combo-box, containing different **access keys** for the AccuSoft products that you bought, and allows you to generate a **Development license** for each access key.

The **Development license** enables you to use any of the ImageGear features on the limited number of computers.

You need to generate a license for each machine where you need to use the product. You can use the **EnvParamsUtility** utility from your ImageGear installation to obtain your **hardware ID**.

NOTE: If the hardware parameters of the machine are changed, you will receive an error (when using the ImageGear) with the following text **"The hardware ID of this machine doesn't match the license information."**

If the hardware parameters of your machine are changed, please log in to the development web page to associate the license with your new hardware ID. Please note that you can use the limited number of the hardware IDs associated with the license. If you reach the limit then you can remove unused hardware IDs from the license.

The product samples are configured to read the license from the file that is defined in the **IMAGE_GEAR_LICENSE_DIR** variable.

NOTE: The **license key** is kept in the file with the name **'accusoft.<solution name>.imagegear'**. The ImageGear installation comes with the **solution name** that is predefined as **'AccuSoft 1-105-15'**. The blank **"accusoft.AccuSoft 1-105-15.imagegear"** license file is included to the installation. The license obtained from the AccuSoft Web site should be inserted into this file. When it is inserted, the variable **IMAGE_GEAR_LICENSE_DIR** has to be defined and should contain the path to the license file (for example **$HOME/AccuSoft**). If the **IMAGE_GEAR_LICENSE_DIR** variable is not defined, ImageGear will look for the license file in the **/usr/lib/** directory.

Please refer to the section ImageGear Licensing API for the details on adding the licensing API calls to your ImageGear application.

NOTE: If you do not have an Internet connection, please call AccuSoft Customer Service at 1-508-351-9092.

NOTE: Development license enables one developer to develop any applications but in order to deploy your ImageGear-based application you need to obtain a **Deployment license**. Please contact AccuSoft when you're ready to deploy your solution.

Getting Started

This chapter helps you to get started with ImageGear for Unix. The ImageGear toolkit contains a variety of sample application programs that you can load at the same time as you install ImageGear's program files. The samples are provided in source form. AccuSoft permits you to use the sample programs' source modules to create and expand your own applications.

The samples included with this software contain a variety of functions that allow you to preview ImageGear's image processing features. They allow users to load an image and manipulate it in several ways.

You can begin developing an application easily by using the code provided in the samples. Choose a sample, use it as a template, and make a backup copy to preserve it. This way you begin with a working program that displays images on your screen from the start. You can easily add, test, and debug each new feature as you add it to your code.

This chapter is set up as a tutorial, and uses the sample applications as a guide. During the tutorial, you will perform a variety of functions using the sample images. You will:

- Convert an image
- Load an image and display it
- Convert an image to DDB format and display it
- Display an image with scrollbars

While performing the steps in this tutorial, you will be introduced to the ImageGear constants that you can use to greatly enhance the flexibility of ImageGear's functions.

To begin with the tutorial, access the samples on your system. The samples are located in the directory to which you have installed ImageGear. (See the sections Directory Structure and Description of Installed Files).

The Console sample application is available via the Command Line only. It does not require or use a User Interface. The Filters sample requires Motif.

NOTE: The **`IG_initialize()`** and the **`()`** functions are UNIX-specific functions and must be called in every imaging application. For more information about these functions please see Chapter "Core Component API Function Reference" .

Converting an Image

This sample program (console.c) demonstrates how to load an image in one format and convert it to another format. To run this program, perform the following steps:

- Run the following command-line parameters:

```
-i[ni] ifilename
```

- Load the ni-th page from the multipage image filename using the **IG_fltr_load_file()** function:

```
nErrCount=IG_fltr_load_file(ifilename, ni, &hIGear);
```

  `IG_fltr_load_file()` loads the image into memory and associates its page with `hIGear`.

  Use the command-line parameters:

```
-o[no] ofilename
```

- Save the already-loaded **hIGear** image into **ofilename** as no-th page using the **IG_fltr_save_file()** function:

```
nErrCount=IG_fltr_save_file(hIGear, ofilename, nFormat, no,
FALSE);
```

  This function saves the image in the format of the `nFormat` parameter value.

- Set the output format value using the command-line parameters:

```
-f format_id
```

  This value is an integer constant taken from ImageGear's master include file **accucnst.h**.

- Use the **igconsole** sample to get a list of possible output formats by issuing the following command:

```
igconsole -p
```

  This command returns a list of all currently available output formats by:

```
<format_id>:<Format_name>.<Compression_name>
```

  The contents of the list depend upon the component that is attached (Please see the section "Using ImageGear Component Manager" on page -187). For example, if the **LZW** component is loaded, then the **GIF/LZW** format is available for use and will appear in the list.

- Use the **-p** command to return the list of all possible output formats for given image. For example, use the command:

```
igconsole -i multpage.tif -p
```

  loads the first page of a multpage.tif file and then returns a list of all available output formats to which the loaded image can be saved.

- Use the following call to exclude all formats that are not applicable to the current loaded format:

```
igconsole -p
```

- Use the following command to attach a component:

```
-e component_filename authentication password
```

  The following example loads a second page from a tiff6.pdf document and saves it to a JPEG file:

```
igconsole -e libIGPDF.so EVAL PASSWORD -i2 tiff6.pdf -f 393237 -
o tiff6_p2.jpg
```

  The following example displays code that converts an image:

```
char   *argv[]

HIGEAR hIGear;/* ImageGear image handle */
AT_ERRCOUNTnErrCount;/* Number of errors that ocurred */
...
/* Load the image in argv[1] */

nErrCount = IG_fltr_load_file(argv[1], 1, &hIGear);
if (nErrCount)
{
    ErrorReport(nErrCount);
    exit(1);
}

/* Save the image as a BMP in argv[2] */

nErrCount = IG_fltr_save_file(hIGear, argv[2], IG_FORMAT_BMP,
0,
    FALSE );
if (nErrCount)
{
    ErrorReport(nErrCount);
    exit(1);
}
...
```

Loading and Displaying an Image

The Filters sample is a fully functional Motif-based GUI sample that demonstrates all major operations that can be accomplished using ImageGear. In this section, we describe how to display an image with the most common parameters by invoking just two ImageGear functions.

Although all ImageGear functions are based on pure **Xlib** and **XToolkit** capabilities and do not require additional libraries, the Filters sample uses Motif 1.2 libraries to demonstrate several advanced ImageGear functions. Motif can be found at http://www.opengroup.org/openmotif/.

The **`IG_dspl_update_rect_set()`** function specifies the rectangular area of a window to be repainted and the **`IG_dspl_update_rect_get()`** function draws the images according to currently set display parameters. These parameters allow you to set various geometric and orientation layouts as well as perform image anti-aliasing, dithering, transparency, and gamma correction. Detailed information about these attributes are described in the section entitled Image Processing.

According to the Motif programming concept, a Callback function, `wndDrawImage_callback`, in the `filters.c` module is called every time a window must be updated. The **`IG_dspl_update_rect_set()`** function passes a pointer to a structure with information that is part of the window to ImageGear. After that, **`IG_dspl_image_draw()`** should be called to make the actual drawing in this area.

Below is a segment from the **`filters.c`** module that demonstrates this:

```
void  wndDrawImage_callback( Widget w, XtPointer data, XtPointer
call_data )
```

```
{
XmDrawingAreaCallbackStruct*cbs =
   (XmDrawingAreaCallbackStruct*)call_data;
XExposeEvent*event = (XExposeEvent*)cbs->event;
AT_RECTANGLE UpdateRect;

...
/* Redisplay only on last event of the series */
if( IG_image_is_valid( hIGear ) )
{
   /* Set the dirty rectangle for repaint */
   UpdateRect.x = event->x;
   UpdateRect.y = event->y;
   UpdateRect.width  = event->width;
   UpdateRect.height = event->height;

   IG_dspl_update_rect_set( hIGear, IG_GRP_DEFAULT, &UpdateRect
);

   /* Repaint window on expose events */
   IG_dspl_image_draw( hIGear, IG_GRP_DEFAULT, XtWindow(w),
       XDefaultGCOfScreen(XtScreen(w)), NULL );
}
...
}
```

Converting an Image to DDB and Displaying It

There are several internal formats used in the ImageGear library to represent an image, and there are groups of functions that allow you to convert it to an **Xlib**-compatible DDB (Pixmap) file, and back to its original format. Creating a DDB displays an image on a pixmap (not on a window). All display-related parameters are used in a DDB display the same way that they are used for displaying an image in a window.

Use the **IG_dspl_DDB_create()** function to create a pixmap object from a **HIGEAR** image function. This function accepts a **HIGEAR** image and the pixmap's width and height as input and returns a pixmap object as output. You can optionally remove the source **HIGEAR** object after creating the object.

To draw a pixmap, use either the standard Xlib functions or ImageGear's **IG_dspl_DDB_draw()** function.

Below are segments from the Filters sample that demonstrate how an image is loaded (`dlg_open.c`):

```
AT_ERRCOUNT dlgOpenLoadPage(LPCHAR lpFileName, UINT nPage,
LPHIGEAR lphIGear)
{
int          fd;
AT_ERRCOUNT  nErrCount;
LPVOID       lpPrivate;
LPFNIG_ERRSTACK_ADD      lpCBFunc;
LPFNIG_ERRSTACK_CLEAR    lpCBFunc2;
```

```
fd = open( (char*)lpFileName, O_RDONLY );
if( fd>0 )
{
    IG_err_callback_get( &lpPrivate, &lpCBFunc, &lpCBFunc2 );
    IG_err_callback_set( NULL, NULL, NULL );

IG_fltr_ctrl_set(IG_FORMAT_PDF, "FILE_NAME",
(LPVOID)szFileName,
                                        strlen(szFileName) + 1);
    IG_err_callback_set( lpPrivate, lpCBFunc, lpCBFunc2 );

    nErrCount = IG_load_FD( fd, 0, nPage, 0, lphIGear );
    close( fd );
}else
{
    nErrCount = 1;
    *lphIGear = (HIGEAR)NULL;
}
return nErrCount;
}
```

and converted to a pixmap and back to **HIGEAR** (**filters.c**):

```
void  mnuView_callback( Widget w, XtPointer data, XtPointer
call_data )
{
AT_DIMENSION        width, height;
int                 item_no = (int)data;
switch( item_no )
{
...
case  2: /* To Pixmap */
    IG_image_dimensions_get( hIGear, &width, &height, NULL );
    IG_dspl_DDB_create( hIGear, IG_GRP_DEFAULT, NULL, width,
height, TRUE, &hPixmap, &hPalette );
    hIGear = (HIGEAR)NULL;
    ImageViewUpdate();
    break;

case  3: /* From Pixmap */
    IG_dspl_DDB_import( NULL, hPixmap, hPalette, &hIGear );
    ImageViewUpdate();
    break;
...
}
}
```

and how to draw a pixmap (**filters.c**):

```
void  wndDrawImage_callback( Widget w, XtPointer data, XtPointer
call_data )
{
XmDrawingAreaCallbackStruct*cbs =
    (XmDrawingAreaCallbackStruct*)call_data;
```

```
XExposeEvent*event = (XExposeEvent*)cbs->event;
AT_RECTANGLE UpdateRect;

if( hPixmap!=NULL )
{
   IG_dspl_DDB_draw( XtWindow(w),
XDefaultGCOfScreen(XtScreen(w)), hPixmap, hPalette, 0, 0 );
}
}
```

Displaying an Image with Scroll Bars

The Filters sample program demonstrates how to add Image scrolling to your application. This example is based on ImageGear's **IG_dspl_scroll_get()**, **IG_dspl_scroll_set()**, and **IG_dspl_scroll_to()** functions. If the size of the displayed image is larger than the viewable area of output device, ImageGear automatically gets and sets the scrolling position of the image within the area. The **IG_dspl_scroll_get()** function returns the minimum, maximum, and current scroll positions in both horizontal and vertical directions. The **IG_dspl_scroll_to()** function allows you to set a new scroll position. Perform the following steps:

1. Create two scrollbars:

```
XtSetArg( args[0], XmNorientation, XmHORIZONTAL );
wHScrollbar = XmCreateScrollBar( main_win, "HScroll", args, 1 );

XtSetArg( args[0], XmNorientation, XmVERTICAL );
wVScrollbar = XmCreateScrollBar( main_win, "VScroll", args, 1 );
```

2. Prepare two Callback functions for scroll-seven handling. These functions should get the new scroll position from the scrollbar widgets and pass it to ImageGear:

```
/
*****************************************************************
*************/
void  ctl_H_Scrollbar_callback(Widget w, XtPointer data,
XtPointer call_data )
{
XmScrollBarCallbackStruct*cbs =
(XmScrollBarCallbackStruct*)call_data;
AT_MODEnScrollMode;
AT_SCROLL_INFO scroll_info;

if( cbs->reason==XmCR_VALUE_CHANGED )
{
   if( IG_image_is_valid(hIGear) )
   {
IG_dspl_scroll_get( hIGear, IG_GRP_DEFAULT, XtWindow(img_win),
&nScrollMode, &scroll_info );

IG_dspl_scroll_to( hIGear, IG_GRP_DEFAULT, XtWindow(img_win),
+ Scroll_info.h_min, scroll_info.h_cur_pos, NULL );
   }
```

```
}
}
```

3. and Vertical scroll-bar

```
/
*****************************************************************
**************/
void  ctl_V_Scrollbar_callback(Widget w, YtPointer data,
YtPointer call_data )
{
YmScrollBarCallbackStruct*cbs =
(YmScrollBarCallbackStruct*)call_data;
AT_MODE            nScrollMode;
AT_SCROLL_INFO     scroll_info;

if( cbs->reason==YmCR_VALUE_CHANGED )
{
   if( IG_image_is_valid(hIGear) )
   {
   IG_dspl_scroll_get( hIGear, IG_GRP_DEFAULT,
YtWindow(img_win),
       &nScrollMode, &scroll_info );

   IG_dspl_scroll_to( hIGear, IG_GRP_DEFAULT,
YtWindow(img_win),
       cbs->value + scroll_info.v_min,
       scroll_info.v_cur_pos, NULL );
   }
}
}
```

4. Associate those Callback functions with scrollbar widgets:

```
XtManageChild( wHScrollbar );
XtManageChild( wVScrollbar );
XtAddCallback( wHScrollbar, XmNvalueChangedCallback,
(XtCallbackProc)ctl_H_Scrollbar_callback, (XtPointer)NULL );
XtAddCallback( wVScrollbar, XmNvalueChangedCallback,
(XtCallbackProc)ctl_V_Scrollbar_callback, (XtPointer)NULL );
```

5. Update the scrollbars every time the image size is changed using the following code:

```
void   ScrollUpdate()
{
AT_MODE nScrollMode;
AT_SCROLL_INFO scroll_info;

if( IG_image_is_valid(hIGear) )
{
   IG_dspl_scroll_get( hIGear, IG_GRP_DEFAULT,
XtWindow(img_win), &nScrollMode, &scroll_info );

   XtVaSetValues( wHScrollbar,
```

```
        XmNminimum, 0,
        XmNmaximum, scroll_info.h_max - scroll_info.h_min + 10,
        XmNsliderSize, 10,
        XmNvalue, (int)(scroll_info.h_cur_pos-
scroll_info.h_min),
            NULL );

    XtVaSetValues( wVScrollbar,
        XmNminimum, 0,
        XmNmaximum, scroll_info.v_max - scroll_info.v_min + 10,
        XmNsliderSize, 10,
        XmNvalue, (int)(scroll_info.v_cur_pos-
scroll_info.v_min),
            NULL );
}else
{
    XtVaSetValues( wHScrollbar,
        XmNminimum, 0,
        XmNmaximum, 0,
        XmNsliderSize, 10,
        XmNvalue, 0,
        NULL );

    XtVaSetValues( wVScrollbar,
        XmNminimum, 0,
        XmNmaximum, 0,
        XmNsliderSize, 10,
        XmNvalue, 0,
        NULL );
}
}
```

Understanding Bitmap Images

This chapter provides an introduction to the workings of the bitmap. It includes descriptions of:

- Pixels
- Channels
- Color Spaces
- 24-bit RGB Images
- 1-bit Images
- 4-bit and 8-bit Images
- Gray-Scale Images
- Color Values Used During Display
- Device-Independent Bitmaps (DIB)
- Device-Dependent Bitmaps (DDBs)
- Vector Images

Pixels

The pixel (or "picture element") is the basic building block of all images displayed by ImageGear. On a display screen, each pixel is a dot. In memory, the pixel is stored as a sequence of bits that determine what color the displayed pixel is. There can also be other data associated with the pixel.

Channels

An image is made up of one or more channels. For each pixel in the image, there are channel values for all of the image's channels. ImageGear supports the following channel types:

- **Color channel** – channel value describes a pixel's color. The number of color channels in an image is determined by the image's color space. For example, an RGB image has three color channels, one for each color component: red, green, and blue. CMYK has four color channels, and grayscale has only one.
- **Alpha channel** – channel value describes how much a pixel should be blended with another pixel during an alpha blending operation. In ImageGear, an image can have only one alpha channel, which can contain either non-premultiplied (as in PNG) or premultiplied (as in TIFF) alpha values. Also, only images with RGB or grayscale color channels can have alpha channels.
- **Extra channel** – extra data associated with the image on a per-pixel basis. For example, if an image being loaded contains multiple alpha channels, the

first alpha channel will be loaded as the image's alpha channel, and the others will be loaded as extra channels. Extra channels are maintained in the image but do not contribute to display or alpha blending operations.

Each channel has a bit depth associated with it. This is the number of bits used to represent a value for the channel. This bit depth can be up to 32 bits on 32-bit platforms and up to 64 bits on 64-bit platforms. It is possible to have an image with multiple channels that differ in bit depth. For example, you could have an RGB image with 8 bits per color channel and a 1-bit alpha channel. Or you could have a 5-6-5 RGB image with 5 bits for red and blue channels and 6 bits for the green channel.

NOTE: Previous versions of ImageGear stored alpha channels as separate, associated images. ImageGear now stores all channel values together for each pixel in an image.

Color Spaces

A color space in ImageGear describes the channels present in an image. The most important part of this description is the color channel configuration. This includes how many color channels there are and how these channels are used to describe colors, which is the usual informal concept of a color space. For example, in the RGB color space there are three channels (red, green, blue) whose values are combined to form colors. In the indexed color space there is one channel which consists of index values into an associated color palette.

In ImageGear, the concept of a color space is extended to also include information about other types of channels besides color, such as alpha and extra channels. An ImageGear color space ID is a bit field which can combine values from the `enumIGColorSpaceIDs` enumeration defined in **`accucnst.h`**.

For example,

- A simple RGB image would be:

  ```
  IG_COLOR_SPACE_ID_RGB
  ```

- A grayscale image with an alpha channel and no extra channels would be:

  ```
  IG_COLOR_SPACE_ID_Gy | IG_COLOR_SPACE_A
  ```

- An RGB image with a premultiplied alpha channel and three extra channels would be:

  ```
  IG_COLOR_SPACE_ID_RGB | IG_COLOR_SPACE_ID_P |
  IG_COLOR_SPACE_ID_Ex
  ```

24-bit RGB Images

24-bit RGB images are images in which each pixel is represented by three 8-bit quantities (thus, 24 bits total) specifying the intensities of red, green, and blue that form the color for the pixel. For example, a pixel that is to be displayed as brightest magenta—magenta being a color formed of equal intensities of red and blue, and no green—would be represented by three bytes having the respective

values of 255, 0, 255 for the red, green, and blue intensities. In RGB representation the brightest white would be (255, 255, 255) and black would be (0, 0, 0).

1-bit Images

Images that contain only one bit per pixel can contain only two colors, since only two pixel values are possible: 0 and 1. The two colors for a 1-bit image are usually black and white, but if the image is to be displayed on an RGB device, the colors are determined by a table called a color palette. A color palette defines the color to be displayed for each possible pixel value. The following table demonstrates the color palette for a 1-bit image, which specifies black for pixel value 0 and white for pixel value 1:

	BLUE	GREEN	RED	Not Used
color 0	0	0	0	0
color 1	255	255	255	255

Each entry in a color palette is a 4-byte structure of type `AT_RGBQUAD`, in which the first three bytes are used to specify the intensities of blue, green, and red respectively, forming the color. The 4th byte is not used. (Note that the ordering in this structure is blue, green, red—not red, green, blue.) `AT_RGBQUAD` and all other structure types mentioned in this chapter are described in detail in the section ImageGear Data Types and Structures.

If instead you wanted to have pixel value 0 displayed as a medium yellow, and pixel value 1 as brightest red, you might use the following color palette (yellow is constructed of equal intensities of red and green, with no blue):

	BLUE	GREEN	RED	Not Used
color 0	0	128	128	0
color 1	0	0	255	0

4-bit and 8-bit Images

A 4-bit image is simply one in which each pixel is represented by 4 bits. Therefore, a 4-bit image can contain 16 (2^4) colors, each pixel having a numerical value between 0 and 15. The color palette for a 4-bit image will therefore normally have 16 entries (0 - 15.) As a 1-bit image might be called a 2-color image, a 4-bit image is also called a 16-color image.

In an 8-bit image each pixel occupies exactly one byte. This means each pixel has 256 (2^8) possible numerical values, from 0 to 255. Therefore, the color palette for an 8-bit image normally contains 256 entries, defining color 0 through color 255. 8-bit or 256-color images are very common because the availability of 256 unique colors provides adequate or even excellent color resolution for most purposes.

Also, when operating upon an image in memory, such as when performing image analysis, transformations, or other image processing, operating on an 8-bit image is much faster than performing the same operations on a 24-bit image. And, of course, an 8-bit image uses only about one-third the memory or file storage space as a 24-bit image.

Because each pixel value of a 1, 4, or 8-bit color image is used as an index into a color palette, these images are sometimes called “indexed color” images. In this manual, 8-bit color images are sometimes referred to as “8i” images, to distinguish them from 8-bit gray level images, which are described below.

Gray-Scale Images

In RGB color representation, (255, 255, 255) results in the brightest white, and (0, 0, 0) results in black. Any time the three intensities are equal, no color is emphasized and the result is a shade of gray. For example (128, 128, 128) would be a medium gray, (240, 240, 240) would be a bright gray approaching white, and (16, 16, 16) would be a dark gray not far from black.

A gray-scale image is one in which the color palette contains only grays, evenly graduated from black (0, 0, 0) for pixel value 0, to white (255,255,255) for the highest possible pixel value. This means that in an 8-bit gray level image, the blue, green, and red intensities for each palette entry are equal to the pixel value. The color palette for an 8-bit gray level image is illustrated below:

	BLUE	GREEN	RED	Not Used
color 0	0	0	0	0
color 1	1	1	1	0
color 2	2	2	2	0
color 3	3	3	3	0
…	…	…	…	…
color 254	254	254	254	0
color 255	255	255	255	0

Some ImageGear image processing operations cannot be performed on 8i (8-bit color) images, because some colors that occur as a result of the processing may not be present in the image’s color palette. For such operations, if the image is 8-bit, it must be 8-bet gray level.

Color Values Used During Display

Although an image's color palette (or in the case of a 24-bit image, each pixel's 24-bit RGB value) normally determines the color to display at each pixel location, there are cases when this is not so. ImageGear maintains a set of Red, Green, and Blue "Look Up Tables" (LUTs), which are used to determine whether the colors to actually display are different from those in the image's palette. The LUTs are modified, for example, if you instruct ImageGear to alter the brightness or contrast of an image. The LUTs can also be set directly by your application (meaning you can display an 8-bit gray-scale image in any 256 colors of your choosing). See the section "Displaying Images" , for further explanation of ImageGear's LUTs, including examples showing how to modify and use them.

The colors displayed for an image may also be modified due to constraints imposed by the display monitor being used. Some display monitors use a single 256-color hardware palette. ImageGear can reload this hardware palette each time an image is to be displayed, but since all images on the screen are being displayed using this one hardware palette, all other images on the screen at the same time will change to reflect the colors of this new palette. For such cases, you can instruct ImageGear as to which image or images are to have precedence in establishing the device's hardware palette. In addition, ImageGear can also inform you when the colors of an image may have changed due to the loading of another image's palette. This is discussed in more detail in the section "Displaying Images" .

Device-Independent/Device-Dependent Bitmaps

A bitmap image, also called a raster image, is an image held in the form of successive rows (called rasters) of pixel data. As already stated, ImageGear's bitmap images use 1, 4, 8, or 24 bits to represent each pixel. But besides these pixel bits, additional information is necessary to describe an image. For example, the width (number of pixels per row) and height (number of rows in the image), are required. For some images, a color palette may be required. Depending upon how the bitmap is to be used, additional information may be required or useful.

To hold bitmap images in memory, ImageGear uses two types of bitmaps, both widely used by many other software products and supported by development platforms such as Microsoft Windows. These are the Device-Independent Bitmap (DIB) and the Device-Dependent Bitmap (DDB). The DIB is by far the more widely used for in-memory operations. It has become the common denominator format because it is so simple: it describes just the image, without reference to characteristics of the device(s) upon which it may later be displayed. While ImageGear provides convenient ways to convert between DIBs and DDBs and to display DDBs, nearly all ImageGear operations are performed on DIBs exclusively.

Device-Independent Bitmaps (DIB)

A DIB consists of a header structure called a **BITMAPINFOHEADER** followed by the color palette, if one is present, followed by the bitmap (pixel) data:



COLOR PALETTE (if any)
BITMAP DATA

The `BITMAPINFOHEADER` structure contains such information as the number of bits per pixel, number of pixels per row, and total number of rows in the image, as well as whether the bitmap data has been compressed for more efficient storage. Its form is shown below. Its fields are described in detail in the section "ImageGear Data Types and Structures".

BITMAPINFOHEADER:	DWORD	biSize
	LONG	biWidth
	LONG	biHeight
	WORD	biPlanes
	WORD	biBitCount
	DWORD	biCompression
	DWORD	biSizeImage
	LONG	biXPelsPerMeter
	LONG	biYPelsPerMeter
	DWORD	biClrUsed
	DWORD	biClrImportant

A color palette is present in the DIB if the image is 1-bit, 4-bit, or 8-bit. The format of the color palette was described at the beginning of this chapter.

The format of the bitmap data is:

- For 1-bit images, the first (leftmost) pixel of a row is held in the Most Significant Bit (MSB) of the first byte of the row. Subsequent bits hold subsequent pixels, at a rate of 8 pixels per byte. The row is padded with zeroes as necessary to assure a multiple of 4 bytes length for each row.
- For 4-bit images, the first pixel of each row is in the 4 Most Significant Bits of the first byte of the row, and each succeeding 4 bits hold each succeeding pixel. As above, the row is zero-padded to a multiple of 4 bytes length.
- For 8-bit and 24-bit images, each pixel of a row occupies exactly 1 or 3 bytes respectively, and again each row is padded with zeroes to a multiple of 4 bytes length. Note that the colors in a 24-bit pixel in a standard DIB are ordered Blue-Green-Red (not Red-Green-Blue).

It should be noted that in a DIB, the first row of the bitmap data is the row to be displayed at the bottom of the image. For historical reasons, many file formats store their bitmap data top row first (this is because most devices to which bitmap data is sent display the rows top-to-bottom. Such devices, which include most CRT display monitors, are often called raster-scan devices.).

Whenever ImageGear loads a file of a format having top-to-bottom row ordering, it automatically reverses the order of the rows, assuring bottom-to-top row ordering for all DIBs, regardless of where the image originated. Keep in mind the ordering of a DIB is therefore “upside down” relative to the convention used in much display software that the top row of a display is row 0 and row numbers increase downward. However, it is important to note, ImageGear’s pixel access functions (such as **`IG_DIB_pixel_set()`**) consider the coordinates 0,0 to refer to the upper left-hand corner of the bitmap data. As with an image shown on the screen, the x values will increase toward the right, and the y values will increase toward the bottom.

See Also:

ImageGear Architecture Diagram.

Device-Dependent Bitmaps (DDBs)

Device-Dependent Bitmaps are bitmaps whose pixel data is organized for convenient dispatch to a particular device or group of devices, such as to a particular type of display monitor or printer. Normally you will not need to concern yourself with DDBs. ImageGear is designed to hold in-memory images in DIBs by default, and to process the image data of DIBs efficiently. But when the speed with which images are displayed is an important factor in your application, keeping large or often-displayed images in memory in DDB format may improve performance. For such cases, ImageGear provides several ways to create or import DDBs, and to display DDBs. Please refer to "Using ImageGear".

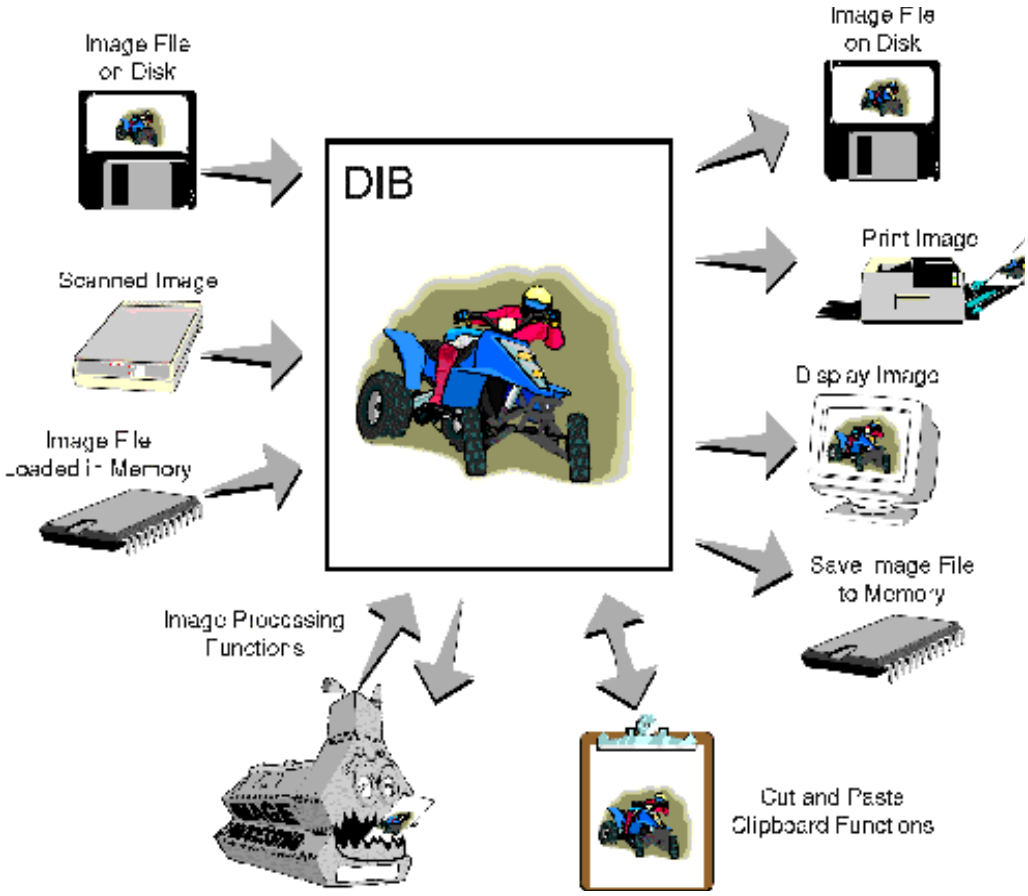
If your application is intensive in the use of in-memory DDB’s, we suggest you refer to Microsoft Windows documentation of Device-Dependent Bitmaps and Device Contexts.

Vector Images

Bitmap images, sometimes also called raster images, have their data organized in horizontal rows of pixels for convenient dispatch to raster-scan devices. Devices and display software also exist that can directly display any line, if given an exact specification of the line. An example is a straight line defined by its two endpoints. Such a line is sometimes called a vector, and an image file whose data is specified exclusively in terms of vectors is called a vector image file.

ImageGear reads most popular formats that contains vector data. While doing so it performs so called “rasterization”, i.e. conversion of vector data into raster representation. For some vector formats such conversion can only be possible for the certain platforms (WMF/EMF vector data can only be rasterized in Windows version of the product, and PICT vector data can only be rasterized in Mac version of the product). For some vector formats (DWG, DXF, HPGL,HPGL/2, DGN) ImageGear also allows to control the projection of 3D vector data into 2D raster bitmap.

ImageGear Architecture Diagram



The diagram in figure 1 gives a conceptual view of the important role played by DIBs in ImageGear. The center image represents a DIB that results when an image is obtained from disk, memory, or a scanned file. Image processing and clipboard functions can be used to alter or merge the actual DIB bitmap data. To the right of the DIB are various routes an image may take after being manipulated by ImageGear: it may be saved to disk or to memory, it may be displayed, or it may be printed.

Using ImageGear

This chapter describes how to use ImageGear's functions to fulfill your applications' imaging needs. The function groups are outlined in ImageGear Overview. There are also many examples that show you how to use ImageGear functionality. Additional examples and recommendations for using ImageGear functions can be found in the API Function Reference chapter for respective component as well as in Getting Started Chapter.

To recognize ImageGear functions naming convention, ImageGear data types and data structure please see the sections:

- Core Component Function Naming Conventions
- How to Call an ImageGear Function
- ImageGear Data Types and Structures

Please read the section ImageGear Licensing API, which describes how to use the ImageGear licensing functions to provide an **evaluation**, **development** or **deployment license** to your ImageGear application.

This chapter includes the following basic sections that describe ImageGear structure and its public functionality:

- Loading Images
- Saving Images
- Displaying Images
- Image Processing
- Pixel Access Operations
- Clipboard operations
- Printing Images
- Thread Safety

The concept of the functions including loading, displaying, saving images, image processing, special effects, thread safety, pixel accessing, etc. are described in these sections.

The following sections describe how to work with special function groups and special groups of images:

- Working with Format Filters
- Working with ImageGear Color Profile Manager
- Working with Global Control Parameters
- Working with Color Space Conversion Engine
- Working with Grayscale Look Up Tables
- Working with Multi-page Images

- Working with Image Utility Functions
- Working with Library Utility Functions
- Working with ImageGear Callback Functions
- Working with Stripped Images
- Working with Tiled Images
- Working with Run Ends Format
- Working with ImageGear Digital Camera RAW Filter

Please see the section Using Format Filters API for Filter Control for the image control parameters setting.

Also take a look at the Non-Image Data Processing section, which describes ImageGear features, providing you with the possibility to read/write metadata information from/in such formats as JPEG, TIFF, EXIF, EXIF-TIFF, EXIF-JPEG, PNG, IPTC, GIF, CIFF, DC RAW.

You can find the detailed information about how to use the functionality of ImageGear components in the following sections:

- Using ImageGear Component Manager
- Using ImageGear ART Component
- Using ImageGear MD Component
- Using ImageGear PDF Component

Core Component Function Naming Conventions

We have carefully named ImageGear Core Component functions with the goal of making them self-documenting and easy to find within Core Component API Function Reference, which is the reference section of this manual.

With the exception of callback functions which always begin with the prefix "`LPFNIG`", the function names adhere to the following conventions:

- All function names begin with the prefix "`IG_`" (always in uppercase). This is to identify it as an **I**mage**G**ear function, and to avoid conflicts with your application's functions.
- Following "`IG_`" is the name of the group to which the function belongs. Table below displays ImageGear function groups and describes their functions

Function name	Function purpose
`IG_clipboard_ …()`	Clipboard
`IG_ … _CB_ …()` or `LPFNIG_ …()_CB_`	Callback
`IG_colorspace_conversion_ …()`	Color space conversion

Function name	Function purpose
IG_comm_ …()	Component manager
IG_cpm_ …()	Color profile manager
IG_DIB_ …()	DIB Info Services
IG_display_ …()	Display
IG_dspl_ …()	Displaying and printing
IG_error_ …(), IG_version_ …(), IG_err_…()	Library utility
IG_fltr_ …()	Format Filter processing
IG_FX_ …()	Special Effects
IG_image_ …() or IG_palette_ …()	Image utility
IG_IP_ …()	Image processing
IG_gctrl_ …()	Global control parameters processing
IG_load_ …()	Loading
IG_mult_ …()	Multimedia
IG_pixel_ …()	Pixel access
IG_save_ …()	Saving
IG_thread_ …()	Thread safety
IG_util_colorspace_ …()	Color space utility

How to Call an ImageGear Function

To call an ImageGear function, use a normal **C** statement, such as:

```
nErrcount = IG_load_file("picture.bmp", &hIGear);
```

The above calls a function named **IG_load_file()**, which loads an image file into memory and provides a handle by which you can refer to the image in subsequent ImageGear function calls.

As described in the "Core Component API Function Reference" , this function requires two arguments: a FAR pointer to a character string specifying the name of the file to load, and a FAR pointer to a variable of type HIGEAR. This second argument will hold the HIGEAR image handle returned by the function. Also, the function returns the number of ImageGear errors, if any, that occurred during execution of the function to the merchant.

Additional statements are needed to support the above function call. These are shown in the more complete example below:

```
#include "gear.h"
   ...
```

```
HIGEAR       hIGear;        /* handle of image in mem */
AT_ERRCOUNT  nErrcount;     /* holds # of errs on stack */
...
nErrcount = IG_load_file (“picture.bmp”, &hIGear);
if (nErrcount != 0) {...}  /* if errs have been reported */
```

The statement `#include “gear.h”` is required in all modules containing ImageGear function calls. It tells the **C** compiler how each ImageGear function must be called, and also defines ImageGear’s special data types, data structures, and constants. The two statements:

```
HIGEAR     hIGear;    /* HIGEAR handle of image  */
AT_ERRCOUNT  nErrcount; /* to check for errs during func */
```

are necessary to define variables used in the function call. The handle of an ImageGear image is always of type `HIGEAR`. The value that is returned by function **`IG_load_file()`** and most ImageGear functions is of type `AT_ERRCOUNT`. This return value contains the number of ImageGear errors generated.

It may be mentioned here that once you have the `HIGEAR` handle of an image, you have the key to the full power of ImageGear. Supplying this handle to an ImageGear function provides all the access it needs to the image’s DIB, color Look Up Tables (LUTs), display attributes, and other information that defines the state of the image in memory. The `HIGEAR` image handle is usually provided to you upon the load, import, or create operation in which the image’s DIB first comes into existence.

The two statements of the example below invoke the ImageGear function **`IG_load_file()`**, and test for any error condition upon its return:

```
nErrcount = IG_load_file (“picture.bmp”, &hIGear);
if (nErrcount != 0) {...} /* if any errs during call... */
```

Most ImageGear functions, including `IG_load_file()`, return a value of type `AT_ERRCOUNT`. This is a count of ImageGear errors reported by the ImageGear function you called and any functions it has called. Whenever this returned count is non-zero, you should examine the contents of the error stack to determine whether some special action is required by your application. How to obtain the contents of the stack, learn specifically which errors have occurred, and clear the stack, is covered in Detecting and Handling Errors.

ImageGear Data Types and Structures

This section provides you with the list of the Data Types and Data Structures that you will encounter in ImageGear `IG_ …()` function call. Each of them has a short description.

Data Types

- `ACCUAPI`: A string directing your C/C++ compiler as to how the given ImageGear function should be called. Remember to include this in your callback function declarations, as shown in the examples of coding callback functions.
- `AT_ERRCOUNT`: Type of the value returned by most ImageGear functions. It is an integer equal to the number of errors placed on the ImageGear error stack during execution of the function and any lower level functions called.

- AT_LMODE: Type of a 32-bit constant (such as IG_SAVE_BMP_RLE), or of a variable containing such a value, used to declare what mode of operation a function should use. Most variables have constants defined for them in accucnst.h. Occasionally a variable of this type will be a Windows constant.
- AT_MODE: Type normally used for a constant that is used to specify an option in a call to an ImageGear function. Examples are IG_CONTRAST_PIXEL or IG_ASPECT_DEFAULT. This type is used when the constant's value can never require more than 16 bits.
- AT_BOOL: An integer that is interpreted as FALSE if 0 and TRUE if non-0.
- AT_DIMENSION: Type usually used for the width in pixels or height in rows, of an image in memory or of a display area on a display device.
- HD: Drawable.
- HBITMAP: Pixmap. .
- HIGEAR: Handle of a comprehensive ImageGear data structure that defines an image along with its display attributes, Look Up Tables, DIB, and other data necessary for maintaining the image and providing access to its pertinent data for your application program.
- HPALETTE: Colormap. .
- HWND: Window ID. .
- LP …: Far (long) pointer to an object of type "…" For example, LPDIMENSION means a FAR pointer to a variable of type
- DIMENSION, LPAT_RECT means FAR pointer to a structure of type
- AT_RECT. FAR is not defined for UNIX, but its use will make future porting efforts easier.
- PIXEL: A BYTE, usually a byte in an image bitmap.
- PIXPOS: A 6432-bit integer value specifying a pixel x or y coordinate.

Data Structures

AT_DIB

Type of the ImageGear DIB header struct :

```
WORD       biSize;
LONG        biWidth;
LONG        biHeight;
WORD        biPlanes;
WORD        biBitCount;
DWORD       biCompression;
DWORD       biSizeImage;
LONG        biXPelsPerMeter;
LONG        biYPelsPerMeter;
DWORD       biClrUsed;
DWORD       biClrImportant;
```

biSize	Total length of this struct in bytes
biWidth	Number of pixels in each raster (row) of the bitmap
biHeight	Number of rasters (rows) in the bitmap
biPlanes	Number of bit planes. Always = 1
biBitCount	Number of bits per pixel (bit depth): 1, 4, 8, 9-16-bit gray level, or 24, 32
biCompression	Type of compression, or 0 (`IG_BI_RGB`) if bitmap not compressed. `IG_BI_RGB`, `IG_BI_RLE`, `IG_BI_CMYK`
biSizeImage	Total number of bytes in bitmap (necessary if bitmap is compressed; may be 0 if bitmap is `IG_BI_RGB`)
biXPelsPerMeter	Pixels per meter horizontally if known (else 0)
biYPelsPerMeter	Rows per meter vertically if known (else 0)
biClrUsed	Number of entries in the color palette that are actually used. (The number of unique pixel values that occur.) If 0, it is assumed all palette entries are used.
biClrImportant	Number of palette entries considered important. If 0, all are important.

AT_POINT

This struct is simply the X and Y coordinates of a point.

```
AT_PIXPOS x;
AT_PIXPOS y;
```

x	X coordinate of the point
y	Y coordinate of the point

AT_RECT

This is the type of all rectangles used in calls to ImageGear `IG_…()` functions

```
AT_PIXPOS left;
AT_PIXPOS top;
AT_PIXPOS right;
AT_PIXPOS bottom;
```

left	X coordinate of the upper left corner of rectangle
top	Y coordinate of the upper left corner of rectangle
right	X coordinate of the lower right corner of rectangle

bottom	Y coordinate of the lower right corner of rectangle

AT_RESOLUTION

This struct describes an image's resolution. This information is used to map between a number of pixels and a physical length measurement. For example, if you have an image which is 600 pixels wide, and the horizontal resolution is specified as 300 DPI (dots per inch, `IG_RESOLUTION_INCHES`), then the image should be 2 inches wide when printed.

```
LONG xResNumerator,
LONG xResDenominator,
LONG yResNumerator,
LONG yResDenominator
AT_LMODE nUnits;
```

See Also:

`IG_image_resolution_get()`

`IG_image_resolution_set()`

AT_RGB

This struct contains three color bytes ordered as in the image bitmap of a standard DIB. Note the order carefully.

```
AT_PIXEL b;
AT_PIXEL g;
AT_PIXEL r;
```

b	Byte containing blue component of color, 0 to 255
g	Green component of color
r	Red component of color

AT_RGBQUAD

Type of a DIB palette entry. This struct contains four bytes; note the order carefully.

```
AT_PIXEL rgbBlue;
AT_PIXEL rgbGreen;
AT_PIXEL rgbRed;
AT_BYTE rgbReserved;
```

rgbBlue	Byte containing blue component of color, 0 to 255

rgbGreen	Green component of color
rgbRed	Red component of color
rgbReserved	Reserved, should be 0

UNIX Platform Specific Functions and Features

Use the following platform-specific functions in ImageGear applications:

- **`IG_initialize()`** - This function initializes the ImageGear library. It must be called before any other ImageGear function is used.
- **`IG_close()`** - When your application no longer needs the use of ImageGear functions, you should close ImageGear using this function. This function frees resources and memory that ImageGear allocates while in use. You should call this function before the application program exits.
- **`IG_dspl_update_rect_get()`** - This function is used to set the dirty area of a window that needs to be repainted. The first parameter is the ImageGear image handle. The second parameter is the display group id indicating where to set the update rectangle. The third parameter is the pointer to actual clipping rectangle. This rectangle is normally set to the rectangle of an expose event.
- **`IG_dspl_update_rect_set()`** - This function returns the update rectangle that was previously set with the function **`IG_dspl_update_rect_set()`**. If the update rectangle was not previously set with **`IG_dspl_update_rect_set()`**, the rectangle returned will be set to the size of the entire drawing surface.

ImageGear Licensing API

Before you can use any ImageGear functionality you need to call the ImageGear licensing API. When you evaluate the product or develop your application using ImageGear you have to call the following function:

```
IG_lic_solution_name_set (LPCHAR lpSolutionName)
```

and pass **'AccuSoft 1-100-15'** as `lpSolutionName` argument.

The ImageGear samples installed with the product already have this call included, so you can simply copy/paste this call from the source code of the samples.

NOTE: ImageGear reads the license information from registry. You have to obtain the product license (evaluation or development) using **AccuSoft License Manager** (see the section ImageGear Licensing) in order for the product to work.

When you're ready to deploy your application please contact AccuSoft about obtaining **License Deployment Kit**. It will explain additional steps (including the API calls) that you need to take in order to deploy your application.

Loading Images

The `IG_load_…()` functions provide you with a convenient way to bring images from image files into ImageGear's sphere of influence. The image files may be on a mass storage device such as a disk, or they may already be in memory.

When you load an image using an `IG_load_…()` function, ImageGear provides you with a handle of ImageGear type `HIGEAR`. Having `HIGEAR` handles for your images, your application has access to ImageGear's full power, and can perform the entire range of ImageGear's imaging operations.

The `IG_load_ …()` functions aren't the only way to get images under ImageGear's control. You can import images that exist as plain bitmaps, DIBs, or DDBs in your application. You can scan images from the outside world, directly into ImageGear. You can locate and extract ASCII or graphics images from multi-page (multi-image) files, using ImageGear's GUI browse and other capabilities.

An `IG_load_ …()` functions create a DIB in memory, and transfer the bitmap and other pertinent information (such as the image's color palette, if one is associated with it, and header information such as width, height, and bits per pixel) into this DIB. These functions do not display the image unless the word "display" is included in the name, such as in **`IG_load_file_display()`**.

Here are a few examples showing how to call the functions in this group. Refer to "Core Component API Function Reference" for more information on the `IG_load_…()` functions.

Example 1:

```
#include "gear.h"
HIGEAR        hIGear;        /* HIGEAR handle returned   */
AT_ERRCOUNT   nErrcount;     /* Count of errors reported */
nErrcount = IG_load_file ("picture.bmp", &hIGear);
```

The above loads file `picture.bmp` from the current directory, creating a DIB in memory, and creates a unique ImageGear handle for it, returning this handle to you in `hIGear`.

Example 2:

```
HIGEAR        hIGear;        /* handle returned by ImageGear  */
char *        lpWhereFile;    /* ptr to image file in mem     */
DWORD         dwWholeSize;  /* size of image file in mem     */
UINT         nPageNum;             /* will be 0 for this call
*/
AT_ERRCOUNT   nErrcount;  /* to test for errors            */

nPageNum     = 0;        /* not a multi-page file        */
lpWhereFile  = ...;     /* where mem image file begins   */
dwWholeSize  = ...;      /* size of whole mem image       */

nErrcount = IG_load_mem (lpWhereFile, dwWholeSize,nPageNum, 0,
&hIGear);
```

IG_load_mem() is used to load an image from memory. The image in memory must be in the same format as if the image were in a disk file. For example it must contain an appropriate header, the bitmap data, and if necessary, a palette. This in-memory file must be of a format recognized by ImageGear (See , "File Format Reference").

In the above call, dwWholeSize must be the size of the entire memory image, not just of the bitmap. lpWhereFile must point to the first byte of this whole image. And nPageNum, if not zero, would specify the page number to load if loading from a multi-page (multiple image) file. Please note that the first page of a multi-page file is page number **1**, not page number **0**. The function IG_load_mem() creates a DIB and loads into it the image you've specified, and returns in hIGear the new HIGEAR image handle by which you will refer to this image in subsequent calls to ImageGear functions.

Names of some IG_load_ …() functions contain the letters "FD", for example **IG_load_FD_CB()**. These functions access the file by its **File Descriptor** handle, which is an integer value returned to you when you open the file using certain Windows functions. You may use these IG_load_FD…() functions to load an image from a file, if the file is already open and your application has its File Descriptor handle. Following is an example showing an ImageGear IG_load_FD…() loading from the file that has been already opened by means of Windows function _lopen() or other file I/O function that returns a File Descriptor handle:

Example 3:

```
HIGEAR        hIGear;      /* handle ret'd by IG_load_FD   */
INT           fd;        /* File Descriptor handle       */
LONG          lOffset;     /* offset to image in file    */
UINT          nPageNum;  /* will be 0 for this call      */
AT_ERRCOUNT nErrcount;    /* to test for errors           */

fd = _lopen ( "picture.bmp", OF_READ );    /* open file     */
nPagenum   = 0;         /* not a multi-page file      */
lOffset    = 0;         /* access file from start     */

/* Load image, and obtain its ImageGear handle: */
nErrcount = IG_load_FD (fd, lOffset, nPageNum, 0, &hIGear);
```

If nErrcount is zero, a DIB is created, the image is loaded, and your HIGEAR object, &hIGear, contains its HIGEAR handle.

TIP: For information about loading CMYK images, see Working With Color Space Options.

See Also:

Using the Format Filters API for Image Loading and Getting Information.

Working With Color Space Options

ImageGear offers different levels of support for the supported color spaces. For CMYK, there are two such levels:

1. your application can read and write CMYK and leave it in its native format
2. your application can read CMYK but converts it to RGB.

ImageGear has two API for working with color space options:

IG_color_space_level_get()	Returns the support level currently set for the color space that you specify.
IG_color_space_level_set()	Sets the support level for the color space you specify.

Each of these functions takes two constants for arguments—one to identify the color space, the second to get/set the desired support level for that color space. The constants are of type `AT_MODE` and can be found in `accucnst.h`.

To select CMYK color space, use the constant `IG_COLOR_SPACE_CMYK`. The constants for CMYK support are `IG_CMYK_CONVERT_TO_RGB`, and `IG_CMYK_FULL_SUPPORT`.

The following call would set CMYK support to “full”, meaning that ImageGear will read in CMYK and allow you to save the image as CMYK:

```
IG_color_space_level_set(IG_COLOR_SPACE_CMYK,
IG_CMYK_FULL_SUPPORT);
```

Saving Images

The `IG_save_…()` family of functions is complementary to the `IG_load_…()` functions. The `IG_save_…()` functions allow you to save images to disk files or to memory, convert files from one file format to another, and to append or insert images as pages to a multi-page file. All `IG_save_…()` functions have the `lFormatType` parameter in common. This allows you to choose which ImageGear-supported file format and compression type (where applicable) to save to. Whereas some functions, such as **`IG_info_get_ex()`**, have separate parameters for the format and compression types, the saving functions have one parameter that covers both.

The Image Rectangle

In ImageGear, an **“image rectangle”** defines the coordinates for the area of the bitmap that will be saved or displayed. The image rectangle represents a rectangular array of pixels from the bitmap that can include the whole image or any rectangular portion of the image.

You can leave the image rectangle set to its default, which includes the whole image, or you can specify a portion of the image by setting the image rectangle with **`IG_dspl_layout_set()`**. A call to this function is included in the example code in Saving an Image to Memory. You can also find a detailed description and example of this function’s use in "Core Component API Function Reference".

TIP: For a more detailed description of the Image Rectangle and other ImageGear display rectangles, see Geometric Layout.

Saving Images to a Disk File

IG_save_file(HIGEAR hIGear, lpszFileName, lFormatType)

is the function normally used to save a HIGEAR image to disk. The name of the file to save to is specified through the second argument, and the format type and compression type (if applicable) in which to save it is specified in the third. If the file format being used supports more than one compression type, more than one constant will be available. The BMP format, for example, is provided with two values for lformatType: IG_SAVE_BMP_UNCOMP and IG_SAVE_BMP_RLE.

The file accucnst.h defines the constants to which lFormatType may be set. These constants are also listed at the end of "File Format Reference".

If the file already exists and the format supports multiple pages, IG_save_file() will append the new image(s) to the file. If the file already exists and it is a single image format type, the file will be overwritten.

Here is an example of code that calls IG_save_file():

```
HIGEAR hIGear;              /* handle ret'd by IG_load_…     */
AT_ERRCOUNT nErrcount;      /* # of errors on stack          */
AT_LMODE lFormatType;       /* format type to save to        */

/*set format and compression to an AT_LMODE constant       */
lFormatType = IG_SAVE_TIF_UNCOMP;

nErrcount = IG_save_file(hIGear, "picture.tif", lFormatType);
```

If you do not know what format your file is in, you may set lFormatType to IG_SAVE_UNKNOWN. ImageGear will check the file's extension and save the image accordingly. If you set lFormatType to other than IG_SAVE_UNKNOWN, ImageGear assumes that you are giving it the correct type and saves the image accordingly.

TIP: Before selecting a file format to save to, you can reference the section entitled ImageGear Supported Bit Depths to ensure that you save to a format that supports the same bit depth as the original image.

Saving to a Disk File Using a File Descriptor Handle

You can use **IG_save_FD()** function to save a HIGEAR image to a file that has already been opened and for which you have a File Descriptor handle (for example, your application may have opened the file using the Windows function _lopen()). IG_save_FD() will permit you to *insert* an image into a multi-page file (instead of merely appending it as **IG_save_file()** would do.)

IG_save_FD() is called with five arguments, of which the HIGEAR image handle and lFormatType have the same meaning as in a call to IG_save_file(). The other parameters are the File Descriptor handle, fd obtained by the Windows function call that opened the file, a reserved argument nReserved, which should always be set = 0, and nPageNum, which allows you to specify what page to insert the image in. If you want to append image(s) to a file, you can set nPageNum to IG_APPEND_PAGE. The following example shows **IG_save_FD()** being used to insert an image as page 3 into an existing a multi-page file:

```
HIGEAR hIGear;/* ImageGear handle         */
INT    d;     /* File Descriptor handle*/
```

```
UINT   nPageNum,/* page # to insert image as*/
       nReserved;/* for future expansion*/
AT_ERRCOUNT nErrcount;/* # of errors on stack*/
AT_LMODElFormatType;/* type of format to save to*/

/ *Windows call to open file with write privileges */
fd = _lopen(“picture.tif”, WRITE);

nPageNum = 3;/* save this image as page #*/
nReserved = 0;* always set to 0 for now*/
lFormatType = IG_SAVE_TIF_JPG; /* format is TIFF-JPEG*/

/* save the HIGEAR image as page 3 of file whose descriptor is
fd:*/
nErrcount = IG_save_FD(hIGear, fd, nPageNum,
nReserved,lFormatType,);
```

If the file “picture.tif” has five pages, the new image will be inserted into position of the third page, and what was formerly page three will now be page four, and so on. Remember that ImageGear treats the first page of a multi-page file as page **1**, not **0**. Setting the `nPageNum` parameter to **0** will append your image(s) to the file. If you set `nFormatType` to a type that doesn't support multiple pages, then the file will be overwritten with a single image.

NOTE: Inserting pages to a multi-page file does not physically rearrange the image in the file. Rather, the offset table is adjusted to hold the new address(es) of the inserted page(s), and the new order of the image within the file.

`lFormatType` should be set to one of the `IG_SAVE_` constants, which also include the compression type. Note that TIFF has more than one compression type, and that there are different constants to represent TIFF with each type of compression. If you call **`IG_save_FD()`** for a multiple-page file, you must supply the format type (from `accucnst.h`) for the image to be appended to the file. If you do not know the format of the destination file, first call **`IG_info_get_FD_ex()`** and use the format information returned to decide which format to use.

The following pseudo-code example demonstrates how to write a multi-page image using `IG_save_FD()`:

```
lFormat = IG_SAVE_TIF_UNCOMP;
fd = _lopen (szFile,OF_READWRITE);
for (i = 1; i<=nPages; i++)
{
/*Seek to the beginning of the file before writing each page
ImageGear will automatically find the correct file position of
the new page */
   seek(fd, 0, SEEK_SET );
   /*Write each next page */
   nErrcount = IG_save_FD(hIGear, fd, i, 0, lFormat);
}
close(fd);
```

NOTE: When you call OS SEEK or an equivalent function to set the file pointer, the stream needs to be positioned at the FIRST byte of a multi-page image. This is the only way in which a filter can recognize that the image is multi-page.

A file should be opened with both **READ** and **WRITE** access.

Saving an Image to Memory

The function **IG_save_mem()** allows you to save a HIGEAR image to memory. The result will be a file image in memory that is identical to the file that would have resulted if you had used any other save function (such as **IG_save_FD()**; the in-memory file image will be given a header, etc. The arguments hIGear and lFormatType have the same meaning as in a call to **IG_save_file()**. However, instead of using a filename to call IG_save_mem() with, you specify the address and size of the memory area to save to. The allocation of memory is discussed further below.

If there already is a valid image file at the address you specify in an IG_save_mem() call, the effect is the same as when using IG_save_FD() to save to an existing file. For example, if the existing image file at the memory address is multi-page, and your lFormatType argument is of the same type, the new image will be inserted or appended to the in-memory file image. As with **IG_save_FD()**, setting nPageNum to IG_APPEND_PAGE will cause the image(s) to be appended to a multi-page file. If the target file's format only supports single images, the memory area will be overwritten with the new single-image file. The previous contents of the memory area will be lost.

Before you call **IG_save_mem()** you need to allocate a memory buffer, and you must supply the size of the allocated buffer to the function. You can determine the appropriate buffer size by making a call to **IG_save_file_size_calc()**. The size returned by this function will include the size of the bitmap data, which can be a portion of the image (the image rectangle) or the whole image, plus any other structures, such as the header or palette. You may need to make additional calls to actually allocate the memory. IG_save_mem() will return the actual size that the file required when it was saved to memory.

The following example code demonstrates how you might use IG_save_mem() to save a file of unknown size:

```
HIGEAR      hIGear;      /* ImageGear handle                */
AT_ERRCOUNT nErrcount;   /* # of errors on stac             */
DIMENSION   nWid, nHi;   /* width & height of image         */
UINT        nBpp,        /* bits per pixel                  */
            nPageNum,    /* page # to insert/append         */
            nReserved,   /* for future expansion            */
            nHorDPI,     /* Horiz res in dots per in        */
            nVerDPI;     /* Ver resolution in DPI           */
AT_LMODE    lFormatType; /* format to save to               */
AT_RECT     rcImageRect; /* Will hold image rectangle       */
DWORD       dwMaxSize,   /* Max #bytes file might need      */
            dwActualSize;/* Actual #bytes it needed         */
HGLOBAL     hBuf;        /* Windows handle of mem area      */
LPVOID      lpBuff       /* Actual address of mem area      */
```

```
lFormatType = IG_SAVE_BMP_RLE;/* Will save to BMP/RLE         */
nPageNum = 0;           /* .BMP not multi-page                */
nHorDPI = 600;          /* Hor resolution = 600 dpi           */
nVerDPI = 300;          /* Hor resolution = 300 dpi           */

/* Get the height and width of the whole image file           */
nErrcount = IG_image_dimensions_get(hIGear, &nWid, &nHi,
&nBpp);

if (nErrcount == 0)       /* if no errors                     */
    {
/* Save only upper leftmost 1" x 1" of image                  */
    rcImageRect.left = 0;
    rcImageRect.top = 0;
    rcImageRect.right = nHorDPI;/* Save exactly 1" of image */
    if ( nWid < nHorDPI )/* unless less than 1" wide        */
       { rcImageRect.right = nWid; }
    rcImageRect.bottom = nVerDPI;/* Save exactly 1" of image  */
    if ( nHi < nVerDPI )  /* unless less than 1" high         */
       { rcImageRect.bottom = nHi; }

nErrcount = IG_dspl_layout_set(hIGear, &rcImageRect);
/*Calculate max # of bytes image rectangle of file may need:
*/
nErrcount = IG_save_file_size_calc ( hIGear, lFormatType,
&dwMaxSize);
if (nErrcount == 0)        /* If no errs alloc sufficient
memory*/
{
   hBuf = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT,
          dwMaxSize);

   /* The mem is movable. Lock it, going to reference it:
*/
   lpBuf = GlobalLock(hBuf);/* lpBuf = present location    */

   /* Now save the image to the mem area:                     */
   nErrcount = IG_save_mem ( hIGear, lpBuf, 0, dwMaxSize,
      nPageNum, nReserved, lFormatType, &dwActualSize);
   if (nErrcount == 0);/* if successful:                     */
   {
   /* Resize the buffer to actual #bytes used:
*/
   hBuf = GlobalReAlloc ( hBuf, dwActualSize, GMEM_MOVEABLE);
   }
GlobalUnlock(hBuf);/* Unlock buffer so moveable           */
}
If (nErrcount) { …}/* Process any errors                   */
```

The program code shown above uses **IG_image_dimensions_get()** to determine the width and height of the image in the unedited file. The user is interested in saving only a 1” x 1” portion from the upper left of the image. If it is smaller than this size, the coordinates of the rectangle will be set to the height and width returned by IG_image_dimensions_get()(the entire image.).

The resolution in dots per inch is needed to convert the width and height from pixels to inches. Based on the horizontal and vertical resolutions, new coordinates are assigned to the members of rcImageRect. Now a call is made to **IG_dspl_layout_set()** to set the image rectangle.

IG_save_file_size_calc() will calculate the maximum size needed to hold the selected rectangular portion of the image in memory. The Windows call GlobalAlloc() allocates dwMaxSize bytes of space for the image. **IG_save_mem()** uses this size to save the image and then returns the actual size of the file in memory. Using the Windows function GlobalRealloc(), the program will now resize the memory held in the buffer to the actual amount needed.

See Also:

Using Format Filters API for Image Saving

Converting Images from One File Format to Another

Converting a file from one file format to another is simply a matter of loading a file from disk using **IG_load_file()** or other load function, and then saving it with a new format type. Be sure, however, that the bit depth of the original file can be supported by the new format type. To help you with this, we have provided you with a Format Bit Depths table in the section entitled ImageGear Supported Bit Depths. Here is an example in which a Windows Bitmap file (BMP) is converted to a TIFF file (.TIF):

```
HIGEAR      hIGear;        /* HIGEAR handle returned        */
AT_ERRCOUNT nErrcount;     /* # of errors on stack          */
AT_LMODE    lFormatType;   /* format to save to             */

/* Set name of BMP file to load and format to save to       */
lFormatType = IG_SAVE_TIF_UNCOMP;

/* Load picture.bmp, obtaining ImageGear handle of image     */
nErrcount = IG_load_file(“picture.bmp”, &hIGear);

if (nErrcount == 0 )          /* If successful:              */
{
    /* save image as a TIFF file:                            */
    nErrcount = IG_save_file(hIGear, “picture.tif”, lFormatType
);
}
```

NOTE: If the file named “picture.tif” already exists, the image will be appended to ‘picture.tif’ because TIFF supports multiple images in a single file.

For more detail on the ImageGear `IG_save_...()` functions, see Core Component API Function Reference.

TIP: In order to save an image using ImageGear, it must have a `HIGEAR` handle. If your image is in DIB format, use: **`IG_image_DIB_import()`** or **`IG_image_create_DIB_ex()`** to assign it a `HIGEAR` handle. If the image is a DDB, use **`IG_dspl_DDB_import()`**.

Displaying Images

ImageGear's family of over 40 display-related functions gives you complete control over where and with what attributes your application displays each of its images. Among the attributes you can set on an image-by-image basis are:

- contrast adjustment
- brightness adjustment
- gamma correction
- dithering
- anti-aliasing enhancement
- rotation
- region within the display area the image is to be displayed (this area is called the Device rectangle)
- the portion of the image to display (this array of pixels within the DIB bitmap is called the Image Rectangle)
- how to fit the Image Rectangle to the Device Rectangle
- the background fill pattern and color to use (for any area of the Device Rectangle left vacant by the image)

Also, by using the image's LUTs (Look Up Tables), you can cause the colors in the image's color palette to be translated to other colors you select (and therefore you can, for example, display an 8-bit gray-scale image in any 256 colors you choose).

All of the above display attributes have their effects during display only. They do not alter either the image bitmap or the color palette in the DIB.

In addition, any time your image is currently being displayed, you can center, zoom, or scroll it from within your application.

How to do all of the above, and how to use additional special purpose `IG_dspl_image_draw...()` functions, is explained in the sections below which discuss basic concepts of displaying images.

Concepts

The key concepts of the new display functionality are display *options,* display *option groups* and display *operations*. A *display option* is a variable that is assigned a value (or a setting) from a predefined list. Every option has a default value associated with it, and therefore, is always defined.

The complete set of all options' values is an *option group*. An *operation* is an action such as displaying or printing an image. Every operation requires an option group whose identifier is passed as a parameter to the function that performs the operation. This group affects the result of the operation and the way in which it is achieved.

Understanding Storage Options

Every handle of an ImageGear image (`HIGEAR`) contains a set of option groups. ImageGear does not impose any restrictions on the size of this set, i.e. on the number of option groups that are stored with any given `HIGEAR`. Do not worry about allocating or de-allocating the storage space for the groups. You can, however, reset all group options to their default values.

Each option group has its own ID. This ID is a `DWORD` integer and is unique in the scope of a given `HIGEAR`, i.e. every pair {`HIGEAR`, `DWORD`} uniquely identifies an option group. The reason for introducing option groups and associating multiple groups with a single image handle is to provide a convenient means of drawing the same image on multiple devices. In this case, a separate option group may be allocated for each of the devices, and later used whenever the image is output to the corresponding device without a need for reassigning options values.

Understanding Display Options

In this section all the options are divided into several categories according to the type of the functionality they affect:

- Geometric Layout
- Dithering, Anti-aliasing, and Palette Handling
- Transparency and Background
- Lookup Tables and Gamma Correction
- Grayscale Look Up Tables

Geometric Layout

The first category includes options that determine the image layout on the destination device:

`ImageRect (AT_RECTANGLE)`	A rectangle that defines the part of the image that is output to the device. It is expressed in image coordinates. By default, this option is set to the entire image.
`ClipRect (AT_RECTANGLE)`	A rectangle that identifies the destination device area that is affected by display operations. Display operations *never* affect parts of the destination device outside of the ClipRect. It is expressed by device coordinates and is initially assigned the entire client area of the destination device.

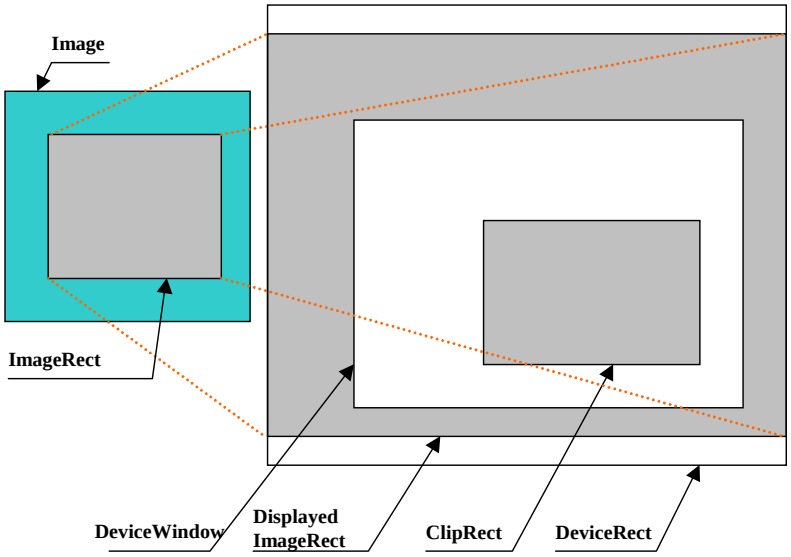
DeviceRect (AT_RECTANGLE)	A rectangle that identifies the area on the destination device upon which the image is projected. Only the portion of the projection falling within the clip rectangle will be seen. The rest of the clip rectangle may be painted with the background color depending on BkMode. The image does not necessarily fit exactly into the device rectangle. The position of the image inside the device rectangle is determined by other options such as AspectMode, FitMode, and AlignMode (see their description below).
DisplayedImageRect (AT_RECTANGLE)	This is not a *display option* that you can set, however this is a very important concept discussed below. This is the rectangular area that represents the image's size and location on the device (and therefore, it is expressed in the device coordinates). It is important to understand that it is not related to the ClipRect, which represents the visible part of the image. The DisplayedImageRect's value can be calculated and used but cannot be directly set.
AspectMode (AT_MODE) and AspectValue (DOUBLE)	These options determine an image's aspect ratio (its width-to-height ratio). AspectMode can be assigned one of two possible values: IG_DSPL_ASPECT_FIXED -- this aspect ratio is the one contained in AspectValue. IG_DSPL_ASPECT_NOT_FIXED -- this aspect ratio is the one of the device rectangle. AspectValue may be any positive number with the following meaning: AspectValue = (DisplayedImageRect.Width / DisplayedImageRect.Height) / (ImageRect.Width / ImageRect.Height). The default values for these options are IG_DISPL_ASPECT_FIXED and 1.0 respectively.
PPMCorrect (BOOL)	This option allows you to take into account the image's resolution when calculating the aspect ratio such that physical width and height are actually used for calculation. If this value is TRUE then ImageRect.Width and ImageRect.Height will be altered based on the horizontal and vertical resolution respectively. Default value is FALSE.
FitMode (AT_MODE)	This value defines how an image fits to the device rectangle while preserving its aspect ratio according to the AspectMode and AspectValue options. The possible values are: IG_DSPL_FIT_TO_DEVICE – in this mode, the image is scaled to fit both the width and height of the device rectangle. IG_DSPL_FIT_TO_WIDTH – in this mode, the image is scaled to fit the width of the device rectangle. IG_DSPL_FIT_TO_HEIGHT – in this mode, the image is scaled to fit the height of the device rectangle. IG_DSPL_ACTUAL_SIZE – in this mode, the device rectangle is ignored and the image is scaled 1:1. The default value is IG_DSPL_FIT_TO_DEVICE.

AlignMode (AT_MODE)	This value defines how the displayed image is aligned relative to the device rectangle. Possible values are bitwise ORs of any two values such that one of them represents the horizontal alignment and the other represents the vertical alignment. The values representing the horizontal alignment are: IG_DSPL_ALIGN_X_LEFT - the image is aligned to the left border of the device rectangle, IG_DSPL_ALIGN_X_CENTER - the image is centered horizontally, IG_DSPL_ALIGN_X_RIGHT - the image is aligned to the right border of the device rectangle. The values representing the vertical alignment are: IG_DSPL_ALIGN_Y_TOP - the image is aligned to the top border of the device rectangle, IG_DSPL_ALIGN_Y_CENTER - the image is centered vertically, IG_DSPL_ALIGN_Y_BOTTOM - the image is aligned to the bottom border of the device rectangle. The default value is IG_DSPL_ALIGN_X_CENTER | IG_DSPL_ALIGN_Y_CENTER.
ZoomMode (AT_MODE)	The value that specifies how the image is zoomed in the horizontal and vertical directions. Possible values for this option are bitwise ORs of any two flags such that one of them represents a horizontal zoom value and the other represents a vertical zoom value. Flags representing the horizontal zoom values are: IG_DSPL_ZOOM_H_NOT_FIXED - this mode allows any horizontal zoom value. IG_DSPL_ZOOM_H_FIXED - in this mode the horizontal zoom factor is taken from ZoomValueH (see below for details). IG_DSPL_ZOOM_V_NOT_FIXED - this mode allows any vertical zoom value. IG_DSPL_ZOOM_V_FIXED - in this mode the horizontal zoom factor is taken from ZoomValueH (see below for details). The default value for this option is IG_DSPL_ZOOM_H_NOT_FIXED | IG_DSPL_ZOOM_V_NOT_FIXED. Please note that AspectMode takes precedence over ZoomMode. In other words, the vertical zoom values are ignored if AspectMode is set to IG_DSPL_ASPECT_FIXED.

ZoomValueH, ZoomValueV (DOUBLE)	These options specify actual horizontal and vertical zoom values according to ZoomMode. Their meaning may be expressed as follows: ZoomValueH = DisplayedImageRect.width / ImageRect.width, ZoomValueV = DisplayedImageRect.height / ImageRect.height. Please note that ZoomValueV is not used if AspectMode is set to IG_DSPL_ASPECT_FIXED.
OrientMode (AT_MODE)	This parameter identifies how the image is oriented before it is drawn on the output device. Possible values are determined by the constants that have the form of IG_DSPL_ORIENT_X_Y, where each of X and Y can be LEFT, TOP, RIGHT or BOTTOM. X stands for the position where the topmost row of the bitmap will be located after applying the transformation and Y stands for the position where the leftmost column of the bitmap will be located after applying the transformation. For example, IG_DSPL_ORIENT_RIGHT_TOP means that the left-most column will become the image's new topmost row and the topmost row will become the image's new rightmost column. The image will be rotated 90 degrees. The following constants are defined: IG_DSPL_ORIENT_TOP_LEFT - the image is displayed unchanged IG_DSPL_ORIENT_LEFT_TOP - the image is rotated 270 degrees and then flipped vertically IG_DSPL_ORIENT_RIGHT_TOP - the image is rotated 90 degrees IG_DSPL_ORIENT_TOP_RIGHT - the image is flipped horizontally IG_DSPL_ORIENT_BOTTOM_RIGHT - the image is rotated 180 degrees IG_DSPL_ORIENT_RIGHT_BOTTOM - the image is rotated 90 degrees and then flipped vertically IG_DSPL_ORIENT_LEFT_BOTTOM - the image is rotated 270 degrees IG_DSPL_ORIENT_BOTTOM_LEFT - the image is flipped vertically The default value is IG_DSPL_ORIENT_TOP_LEFT. Note that when changing an image's orientation, the image rectangle's orientation is also changed. The other rectangles and options remain unchanged.

ScrollbarMode (AT_MODE)	Scrolling is automatically supported in both horizontal and vertical directions. Scroll parameters are automatically calculated by ImageGear. All scroll parameters are stored apart from scrollbars, and therefore, the image can be scrolled in any possible way. If you want to manage the scrollbars yourself, you may do so by using this option. Possible values for this option include any combination of two flags, such that one of them is a “horizontal” flag and the other is a “vertical” flag. Horizontal flags are: IG_DSPL_HSCROLLBAR_AUTO - allows ImageGear to show and hide the horizontal scrollbar depending on the scroll range. IG_DSPL_HSCROLLBAR_ENABLE - in this mode, ImageGear always shows the horizontal scrollbar even if the scroll range is 0. IG_DSPL_HSCROLLBAR_DISABLE - in this mode, ImageGear ignores the horizontal scrollbar and does not set any of its properties. IG_DSPL_VSCROLLBAR_AUTO - allows ImageGear to show and hide the vertical scrollbar depending on the scroll range. IG_DSPL_VSCROLLBAR_ENABLE - in this mode, ImageGear always shows the vertical scrollbar even if the scroll range is 0. IG_DSPL_VSCROLLBAR_DISABLE - in this mode, ImageGear ignores the vertical scrollbar and does not set any of its properties.
ScrollPosH, ScrollPosV (LONG)	These options determine how DisplayedImageRect is moved in the horizontal and vertical directions. These values should fall within the corresponding scroll range. Both are set to 0 by default.
MapMode (DWORD), Viewport (AT_RECTANGLE), Window (AT_RECTANGLE)	These options are device dependent and allow you to set logical coordinates. It is assumed by ImageGear that all coordinates except **ImageRect** are logical. These options are necessary to properly convert logical to device coordinates. For the Windows platform the value of MapMode can be any value accepted by the GDI’s functions GetMapMode/ SetMapMode. The Viewport and Window contain values that are the same as GDI’s SetViewportOrgEx/SetViewportExtEx and SetWindowOrgEx/SetWindowExtEx. The default value is the same as current desktop map mode.

This figure demonstrates the meaning of the rectangles listed above:



Some of the options listed above conflict unless priorities are defined. Consider the general algorithm of display rendering. It consists of several steps, and on each step ImageGear processes some options to introduce modifications to the resulting image.

- ImageRect is oriented according to `OrientMode`. `ClipRect` and `DeviceRect` are calculated according to their definitions above.
- If `FitMode` is set to `IG_DSPL_ACTUAL_SIZE` then `DisplayedImageRect.width` and `DisplayImageRect.height` are set to `ImageRect.width` and `ImageRect.height` respectively. Otherwise, `DisplayedImageRect` is computed using `DeviceRect`.
- According to `ZoomMode`, `ZoomValueH` and `ZoomValueV`, `DisplayedImageRect.width` and `DisplayedImageRect.height` are modified as follows:

```
if( (ZoomMode&IG_DSPL_ZOOM_H_FIXED) != 0 )
    DisplayedImageRect.width =
DisplayedImageRect.width*ZoomValueH,
```

and

```
if( (ZoomMode&IG_DSPL_ZOOM_V_FIXED) != 0 )
    DisplayedImageRect.height =
DisplayedImageRect.height*ZoomValueV.
```

If `AspectMode` is set to `IG_DSPL_ASPECT_FIXED`, then `ZoomValueV` is not used and the above procedure is changed in the following way:

```
If( (ZoomMode&IG_DSPL_ZOOM_H_FIXED) != 0 )
{
DisplayedImageRect.width = DisplayedImageRect.width*ZoomValueH,
```

```
DisplayedImageRect.height =
DisplayedImageRect.height*ZoomValueH.
}
```

- `DisplayedImageRect.x` and `DisplayedImageRect.y` are computed so that `DisplayedImageRect` is aligned as specified by `AlignMode`.
- The scrolling range is calculated so that `DisplayedImageRect` can be viewed inside of `ClipRect`. Then `DisplayedImageRect.x` and `DisplayedImageRect.y` are shifted according to the current scroll position.

Dithering, Anti-aliasing, and Palette Handling

ImageGear supports automatic dithering of source images. This functionality is affected by the following options:

`DitherMode (AT_MODE)`	Can be assigned the following values:
`IG_DSPL_DITHER_AUTO`	Specifies that the destination device color resolution should be used for dithering. In this mode IG automatically applies dithering only when it is necessary.
`IG_DSPL_DITHER_TO_8 BPP`	This mode forces ImageGear to assume that the output device is 8 bits per pixel and perform the necessary dithering.
`IG_DSPL_DITHER_TO_4 BPP`	This mode forces ImageGear to assume that the output device is 4 bits per pixel and perform the necessary dithering.
`IG_DSPL_DITHER_TO_1 BPP`	This mode forces ImageGear to assume that the output device is 1 bit per pixel and perform the necessary dithering.
`IG_DSPL_DITHER_NONE`	Disables ImageGear's dithering. In this mode dithering is performed by the operating system or the device driver.

The following flags can be used with any of the above modes:

`IG_DSPL_DITHER_FIXED_ PALETTE`	If this flag is set, ImageGear will try to use the standard palette when performing dithering. This may be useful if the output device contains more than one image and by using this flag it is possible to draw images with the same palette.
`PALETTE`	This flag is applicable only if the output device is 8 bits per pixel. It tells ImageGear to use the 216 entries Netscape palette. The default value for `DitherMode` is `IG_DSPL_DITHER_AUTO`. Currently only ordered dithering is implemented.

IG_DSPL_ANTIALIAS_MODE (AT_MODE)	A bitmask used to isolate black and white antialiasing modes: ■ IG_DSPL_ANTIALIAS_NONE - anti-aliasing is not used. ■ IG_DSPL_ANTIALIAS_SCALE_TO_GRAY - the scale to gray algorithm is used and the output image becomes 4 bits per pixel ■ IG_DSPL_ANTIALIAS_PRESERVE_BLACK - this mode means ImageGear will try to preserve black pixels while scaling. ■ IG_DSPL_ANTIALIAS_PRESERVE_WHITE - This mode means ImageGear will try to preserve white pixels while scaling. Together with these modes the following flag can be used: ■ IG_DSPL_ANTIALIAS_SUBSAMPLE - specifies ImageGear to use sub-sampling during anti-alias scaling. The output is as good but the speed is greater.
IG_DSPL_ANTIALIAS_RESAMPLE_MODE	A bitmask used to isolate resampling modes: ■ IG_DSPL_ANTIALIAS_RESAMPLE_BILINE - resample with bilinear interpolation
IG_DSPL_ANTIALIAS_COLOR_MASK	A bitmask used to isolate color antialiasing modes: ■ IG_DSPL_ANTIALIAS_COLOR - color antialiasing
AliasThreshold (UINT)	Threshold integer value from 0 to 100. Its meaning depends on the AliasMode value If AliasMode is set to IG_DSPL_ANTIALIAS_SCALE_TO_GRAY then AliasThreshold determines how many black and white pixels are involved in the destination gray pixel value. Value 100 causes ImageGear to take 100% white pixels, value 0 causes ImageGear to take 100% of black pixels, and the default value is 50 which means 50% of white and 50% of black pixels. If AliasMode is set to IG_DSPL_ANTIALIAS_PRESERVE_BLACK, then AliasThreshold determines how many black pixels should be preserved. Value 100 means that 100% of black pixels are preserved. The default value is 50. If AliasMode is set to IG_DSPL_ANTIALIAS_PRESERVE_WHITE, then AliasThreshold determines how many white pixels should be preserved. Value 100 means that 100% of white pixels are preserved. The default value is 50.

DevicePalette (HPALETTE)	This option is a palette in OS-dependent format. This palette is implemented before drawing pixels onto the output device. By default, this option is NULL, which means that it is created every time; either from the logical palette of the image, or from the logical palette used in dithering. If this option is not NULL then it should be a valid palette handle. ImageGear does not perform logical processing of the specified palette; it only implements it before drawing. Incorrect usage of this option may distort the image on the destination device.
PaletteMode (AT_MODE)	This option specifies how to use the DevicePalette option when the destination device does support palette operations. Possible values are: IG_DSPL_PALETTE_HIGH – tells ImageGear to use the palette in the high priority mode. This means that the operating system palette manager will try to best map colors of DevicePalette to the system palette. IG_DSPL_PALETTE_LOW – tells ImageGear to use the palette in the low priority mode. In this mode the palette manager will try to best preserve current view of the destination while drawing the new image on it. IG_DSPL_PALETTE_DISABLE – tells ImageGear not to implement DevicePalette in the destination device while drawing the image.

Transparency and Background

ImageGear display functionality supports transparency by **color** and by **mask**. The former means that it is possible to draw images with the specified color being transparent. The latter allows providing a one bit per pixel image and using it as the transparent mask. There are a few options related to transparency and background support:

TranspMode (AT_MODE)	Option that contains transparency flags. If its value is IG_DSPL_TRANSPARENCY_NONE then transparency is disabled. Each of the next flags enables some transparency-related features:
IG_DSPL_TRANSPARENCY_COLOR	If this flag is set then the transparent color is enabled and the color's value that is assigned to the TranspColor option is used as transparent while drawing the image.
IG_DSPL_TRANSPARENCY_MASK	If this flag is set then the transparency mask is enabled and the TranspMask option is used as the transparent bitmap.

IG_DSPL_TRANSPMASK_ STRETCH_TO_IMAGE	This flag is used when the transparency mask is enabled. If this flag is set then the TranspMask image is resized and oriented along with the image being displayed. In other words, if during display operation the image is scaled by factors DX and DY in the horizontal and vertical directions respectively, then the TranspMask image is scaled by the same factors. If this flag is not set then the transparency mask is not scaled. This flag is ignored unless the IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE flag is set.
The next three flags are exclusive so that only one of them can be set at a time:	
IG_DSPL_TRANSPMASK_ LOCATE_TO_IMAGE	This flag is applicable if the transparency mask is enabled. If this flag is set then the mask is calculated in the image-dependent coordinate system. This means that the mask is oriented as the image is and the MaskLocation option is calculated from the original image's ImageRect. If this flag is not set then the IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE flag is ignored.
IG_DSPL_TRANSPMASK_ LOCATE_TO_CLIPRECT	This flag locates the transparent mask relative to the ClipRect, so that the MaskLocation is calculated from this rectangle's left top point.
IG_DSPL_TRANSPMASK_ LOCATE_ABSOLUTE	This flag locates the transparent mask on the output device so that the mask's left top point of has the coordinates specified in the MaskLocation option. The default value for TranspMode option is IG_DSPL_TRANSPARENCY_NONE which means that transparency is disabled.
TranspColor (AT_RGB)	This (RGB) triple specifies the transparent color and it is used if the IG_DSPL_TRANSPARENCY_COLOR flag is set in TranspMode (see its description above). If the transparent color is enabled, then all the pixels in the image that have values equal to TranspColor are drawn as transparent.
TranspMask (HIGEAR)	This option specifies the transparency mask and it is used if the IG_DSPL_TRANSPARENCY_MASK flag is set (see the TranspMode's description above). This is a normal HIGEAR image, and its location on the screen depends on other flags described above and the MaskLocation option. The application code is responsible for creating and deleting this mask.
MaskRect (AT_RECTANGLE)	This rectangle specifies which part of the TranspMask image should be used as the transparency mask. It is an analog of the original image's ImageRect but it applies to the TranspMask image. By default, it is initialized with the empty rectangle, which means the complete TranspMask image will be used.

MaskLocation (AT_POINT)	This option specifies how the transparency mask is located relative to either the image or the device d epending on the flags set in TranspMode.
BkMode (AT_MODE)	This option specifies how ImageGear should fill the area of ClientRect that is not covered by the image's pixels. If its value is IG_DSPL_BACKGROUND_NONE then the background is disabled and IG does not fill this area. Other possible values are as follows: IG_DSPL_BACKGROUND_UNDER_IMAGE – if this flag is set, then the image's transparent pixels are drawn with current BkColor and BkBrush (see their description below). The area outside of DisplayedImageRect is not affected. IG_DSPL_BACKGROUND_BEYOND_IMAGE – if this flag is set, then the transparent pixels that are outside of DisplayedImageRect are drawn with current BkColor and BkBrush (see their description below). The default value for this option is IG_DSPL_BACKGROUND_UNDER_IMAGE| IG_DSPL_BACKGROUND_BEYOND_IMAGE.
BkColor (AT_RGB)	This option is a (RGB) triple that specifies the background color. This color is used to fill the area of ClipRect that is not covered by the image's pixels.
BkBrush (HBITMAP)	This option is the handle of the bitmap that stores the brush to be used. The application code is responsible for creating and deleting this brush.

Lookup Tables and Gamma Correction

ImageGear supports gamma correction preprocessing before the image is drawn onto the destination device. This operation does not affect the image itself but only changes its appearance.

There are three options that allow the controlling of the gamma correction

- **RedLut** (LPBYTE) - this is a 256-entry array of bytes that contains a new value for each of the possible 256 intensities of the red color.
- **GreenLut** (LPBYTE) - this is a 256-entry array of bytes that contains a new value for each of the possible 256 intensities of the green color.
- **BlueLut** (LPBYTE) - this is a 256-entry array of bytes that contains a new values for each of the possible 256 intensities of the blue color.

Each of the three option's default value is the identity array.

NOTE: If the source image is not in the RGB color space then it is first converted to the RGB color space and then all the lookup tables are applied.

There are high level options, which automatically create all necessary lookup tables.

- **Contrast**, **Brightness**, **Gamma** (`DOUBLE`) – these three options are actually parameters which specify how to calculate the lookup tables to get the necessary color effects.
- **Parameter Contrast** specifies the contrast level to produce. Values greater then 1.0 increase contrast, values less than 1.0 decrease contrast. Values less than 0.0 will invert contrast (exchange dark and light).
- **Parameter Brightness** specifies the brightness adjustment. Possible values range from –255.0 to +255.0.
- **Parameter Gamma** controls the non-linear contrast adjustment. Values greater then 1.0 increase contrast, values less than 1.0 decrease it. Usual range is from 1.8 to 2.2

The default values are:

- Contrast = 0.0,
- Brightness = 1.0,
- Gamma = 1.0

Grayscale Look Up Tables

The ImageGear display API allows storing a grayscale (single-channel) look up table with an image's display settings. The grayscale look-up table can be set for any image, but ImageGear only uses it with 8...16-bit grayscale images, and ignores it with the other images. This look up table specifies a transform from 16-bit image to 8-bit image, which allows to display a particular part of 8...16-bit image's contrast range, or enhance image's contrast.

A grayscale LUT can be stored with a 16g image as well. However, if both the image and its display contain LUTs, the display LUT overrides the image's LUT. A LUT can be removed from an image and attached to a display, or vice versa.

Storing a grayscale LUT with display settings allows you to display the same image with different LUTs simultaneously, in different windows.

A grayscale LUT can be used in combination with a RGB LUT. The grayscale LUT is applied first, and then the RGB LUT is applied.

Use **`IG_dspl_grayscale_LUT_update_from()`** to create or update a grayscale LUT with the specified LUT. LUT data will be copied to the display settings. Set the `lut` parameter to NULL to remove grayscale LUT from display settings.

Use **`IG_dspl_grayscale_LUT_exists()`** to check whether display settings contain a grayscale LUT.

Use **`IG_dspl_grayscale_LUT_copy_get()`** to obtain a copy of the display grayscale LUT.

The ImageGear Medical component provides a set of API functions that allow building grayscale LUTs according to various DICOM display settings.

The only allowed LUT configuration for display is: InputDepth = 16, OutputDepth = 8, Output is unsigned. Such LUTs can be used with grayscale images which depth is 8… 16 bits per pixel.

See Also:

Working with Grayscale Look Up Tables

Image Processing

ImageGear's comprehensive family of image processing functions permits you to perform both simple and complex image-modifying operations using a single function call. Image alterations such as contrast enhancement, sharpness adjustment, color reduction/promotion, image merging ("blending"), and "special effects" are performed using the functions in this group.

NOTE: Image processing functions are always named beginning with `IG_IP_ …()` or `IG_FX_ …()`. Be careful not to confuse these functions with `IG_dspl_image_draw …()`functions designed to perform similar operations during displaying images. The `IG_dspl_image_draw …()` functions affect only how the image will be displayed. The image processing functions actually alter the pixel data in the DIB image bitmap, or the DIB palette, or both.

Many image processing functions permit you to specify a rectangular region within your image, limiting the function's operation to that region. In such cases, the address of an `AT_RECT` structure is supplied as an argument. (Specify this argument as `NULL` to operate on the whole image.)

NOTE: The "image rectangle" setting made by calls to function **`IG_dspl_layout_set()`** is not used by the image processing functions.

In this section, ImageGear's image processing functions are grouped as shown below. You may want to refer to the individual discussions of the groups as they become pertinent to your application's development needs.

- Color Reduction
- Color Promotion
- Contrast Alteration
- Image Transformation
- Image Analysis
- Special Effects
- Blending and Combining Images
- Image Correction

- Image Maintenance

If you encounter an occasional image processing term that you are not familiar with, be sure to refer to the comprehensive "Glossary" . Also, refer to the functions entries in "Core Component API Function Reference" for the detailed calling sequences and additional information and examples. In a function's entry, be sure to check the "Bits Per Pixel" line. It tells which of bit depths may be processed using that function. The descriptions of the different bit depths (1, 4, 8i, 8-bit gray level, 9-16-bit gray level, 24, and 32) and how they are stored internally in the DIB image bitmap in memory, can be found in "Understanding Bitmap Images".

NOTE: A number of image processing functions can process 8-bit gray level and 24-bit images, but cannot process 8i (8-bit indexed color) images. This is because in 8i images the pixel value does not itself describe the pixel (its color or its intensity), but is merely an index into the palette. If you want to call such a function for an 8i image, first promote the image to 24-bit using function **`IG_IP_color_promote()`**.

NOTE: The following types of image processing can be performed on an image that is loaded with its CMYK data intact (not translated to RGB): **`IG_IP_crop()`**, **`IG_IP_flip()`**, **`IG_IP_resize()`**, **`IG_IP_rotate_multiple_90()`**, **`IG_IP_rotate_any_angle()`**, **`IG_IP_contrast_invert()`**, **`IG_IP_contrast_stretch()`**, and **`IG_IP_contrast_equalize()`**. The rest of image processing API can be used after conversion of CMYK image to RGB format.

Color Reduction

Color reduction in general results in an image with fewer colors than the original, even if the bit depth is not changed. There are a number of color reduction methods and ImageGear provides you with several different functions from which you can choose the one that is most suitable in a given case. Below you can see the color reduction methods that are available, and the functions that perform them:

`IG_IP_color_reduce_bayer()`	Reduces a 4, 8, or 24-bit image to a 4-bit image using a Bayer dithering algorithm.
`IG_IP_color_reduce_to_bitonal()`	Reduces a 4, 8, or 24-bit image to a 1-bit or "bitonal" image.
`IG_IP_color_reduce_median_cut()`	Reduces a 24-bit image to 8 bits using the median cut algorithm.
`IG_IP_color_reduce_diffuse()`	Reduces a 4, 8, or 24-bit image to a 1-bit or 4-bit image using a diffusion algorithm.

IG_IP_color_reduce_popularity()	Reduces a 24-bit image to an 8-bit image, while preserving its most prevalent or popular colors.
IG_IP_color_reduce_octree()	Reduces a 24-bit or 8-bit image to an 8-bit or 4-bit image. Uses an efficient algorithm that gives a result as near the original as possible, using the number of colors you specify.
IG_IP_color_reduce_halftone()	Reduces a 4, 8, or 24-bit image to a 1-bit image using a halftone pattern.

Refer to the function descriptions in Core Component API Function Reference for the calling sequences and further information on the above.

The additional function **IG_IP_convert_to_gray()** is also considered a color reduction function. It is called as follows:

```
IG_IP_convert_to_gray ( hIGear );
```

It always converts the image to 8-bit gray level. The resulting DIB has a 256-entry palette, and each 8-bit pixel value in the resulting image bitmap is a weighted average of the three color intensities of the pixel in the original image.

Color Promotion

Color promotion is the process of increasing the bit depth, or number of bits per pixel, of an image. The color of each pixel is retained. ImageGear provides one function that handles promotion to any bit depth. It is called as follows:

```
IG_IP_color_promote ( hIGear, IG_PROMOTE_TO_24 );
```

The constant shown in the example above may instead be `IG_PROMOTE_TO_4` or `IG_PROMOTE_TO_8`. Your image must of course originally have fewer bits per pixel than the bit depth to which you are promoting it.

When promoting to 4 or 8 bits, the promotion is accomplished by simply increasing the number of bits per pixel for each pixel (but without changing the pixel's value), and increasing the size of the DIB palette (the added palette entries are set to all zeroes).

Function name	Bpp In	Use or create palette for output image	Bpp Out
IG_IP_color_reduce_bayer()	24	use standard B/W palette	1
IG_IP_color_reduce_bayer()	8	use standard B/W palette	1
IG_IP_color_reduce_bayer()	4	use standard B/W palette	1
IG_IP_color_reduce_bayer()	24	use given palette	4
IG_IP_color_reduce_bayer()	8	use given palette	4
IG_IP_color_reduce_bayer()	24	use standard palette	4
IG_IP_color_reduce_bayer()	8	use standard palette	4

IG_IP_color_reduce_diffuse()	24	use standard B/W palette	1
IG_IP_color_reduce_diffuse()	8	use standard B/W palette	1
IG_IP_color_reduce_diffuse()	4	use standard B/W palette	1
IG_IP_color_reduce_diffuse()	24	use given palette	4
IG_IP_color_reduce_diffuse()	8	use given palette	4
IG_IP_color_reduce_diffuse()	24	use standard palette	4
IG_IP_color_reduce_diffuse()	8	use standard palette	4
IG_IP_color_reduce_halftone()	24	use standard B/W palette	1
IG_IP_color_reduce_halftone()	8	use standard B/W palette	1
IG_IP_color_reduce_halftone()	4	use standard B/W palette	1
IG_IP_color_reduce_median_cut()	24	create optimal palette	8
IG_IP_color_reduce_octree()	24	use given palette (maxcolors > 16)	8
IG_IP_color_reduce_octree()	8	use given palette (maxcolors > 16)	8
IG_IP_color_reduce_octree()	24	create optimal palette (maxcolors > 16)	8
IG_IP_color_reduce_octree()	8	create optimal palette (maxcolors > 16)	8
IG_IP_color_reduce_octree()	24	use given palette (maxcolors <= 16)	4
IG_IP_color_reduce_octree()	8	use given palette (maxcolors <= 16)	4
IG_IP_color_reduce_octree()	24	create optimal palette (maxcolors <= 16)	4
IG_IP_color_reduce_octree()	8	create optimal palette (maxcolors <= 16)	4
IG_IP_color_reduce_popularity()	24	create optimal palette	8
IG_IP_color_reduce_to_bitonal()	24	use standard B/W palette	1
IG_IP_color_reduce_to_bitonal()	8	use standard B/W palette	1
IG_IP_color_reduce_to_bitonal()	4	use standard B/W palette	1

When promoting to 24 bits, the 24-bit color of the pixel (obtained from the image's original palette) becomes the 24-bit pixel in the resulting promoted image. The DIB palette is deleted, since a 24-bit image does not have a DIB palette.

NOTE: The function **IG_IP_convert_to_gray()**, though it could increase the number of bits per pixel in an image, is considered a color reduction function, because it reduces the colors to shades of gray.

Contrast Alteration

ImageGear's contrast alteration functions operate by altering the range of pixel intensities that occur in your image, or by redistributing the occurrence frequency of the pixel intensities. There are five IG_IP_contrast_…() functions, which operate as follows:

IG_IP_contrast_adjust()	Adjusts the contrast of the image by stretching or compressing the range of intensities that occur. Also adjusts brightness, by adding or subtracting a constant you specify, to each intensity value.
IG_IP_contrast_stretch()	Adjusts the contrast of the image by stretching the range of intensities that occur, such that the least intense pixel becomes full black and the most intense becomes full white.
IG_IP_contrast_equalize()	Adjusts the contrast of the image by stretching or compressing sub-ranges of intensities that occur, so that there are an approximately equal number of pixels in each sub-range. This can bring out subtle changes in contrast when contrast is poor in the original, such as in x-ray images.
IG_IP_contrast_gamma()	Adjusts the contrast of the image non-linearly, using an algorithm that tends to correct for the non-linear response of display monitor phosphors, video camera photoreceptors, and photographic emulsions.
IG_IP_contrast_invert()	Inverts each pixel intensity or color, resulting in an image that is a "negative" of the original.

Each of the above functions can operate on any specified rectangular portion of your image. However, when operating on a whole image, each function above can achieve its effect by altering the image's palette instead of by altering its pixel values. An example of a call to an IG_IP_contrast_ …() function is:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_RECT rcRect; /* rectangle to operate on */
IG_IP_contrast_equalize ( hIGear, &rcRect, IG_CONTRAST_PIXEL);
```

If you specify IG_CONTRAST_PALETTE instead of IG_CONTRAST_PIXEL, note that your rectangle argument will be ignored, and the operation will be performed on the whole image.

Several additional functions help you to highlight contrast boundaries:

IG_IP_edge_map()	Produces a resultant image that shows where there are contrast changes in the original image. An area in which there are no contrast changes is black in the resultant image; the stronger the contrast change, the brighter the result. The result tends to have bright lines where there are sharp contrast changes.
IG_IP_pseudocolor_small_grads()	Colors an 8-bit gray level image according to the local rate of p ixel value change. This can be set to expose even very small gradients in brightness.
IG_IP_pseudocolor_limits()	Allows you to block out, to a single color, all pixels below (or above) a given pixel value. This can be used to highlight the portion of an image that is relevant (then **IG_IP_contrast_stretch()** might be called to enhance its contrast), or may be used to see what portions of the image are saturated or unsaturated.

Refer to the descriptions of all of the above in "Core Component API Function Reference".

Image Transformation

An image transformation function is one in which a mathematical algorithm is applied to transform each pixel to a new location or value. Rotation is a good example of a simple image transformation. Image transformation differs from contrast adjustment (which might also be carried out by applying an algorithm) in that the contrast adjustment is trying to achieve some visual improvement or enhancement of the image. A transformation's object is simply to achieve the transformation (e.g., rotate the image).

ImageGear provides the following image transformation functions. Refer to the description of each in "Core Component API Function Reference" for detailed calling sequences as well as other related information:

IG_IP_transform_with_LUT()	Transforms an image by mapping each pixel value through a Look Up Table that you supply to obtain the pixel's new value.
IG_IP_convolve_matrix()	Convolves the 8-bit gray level or 24-bit image using a user-defined convolution kernel.
IG_IP_rotate_multiple_90()	Rotates your image 90, 180, or 270 degrees around its center.
IG_IP_rotate_any_angle()	Rotates your image through any angle you specify around its center.

IG_IP_sharpen()	Causes the dark side of a contrast boundary to become darker and the bright side to become brighter. This makes the image appear sharper. You can control the degree of sharpening applied.
IG_IP_smooth()	Removes graininess in an image, tending to soften or smooth its appearance. You control the degree of smoothing.
IG_IP_flip()	Flips your image right-for-left or top-for-bottom. This is equivalent to rotating the image about a vertical or horizontal axis (respectively) drawn through its center.
IG_IP_resize()	Re-scales the image, changing the size of the image bitmap.

Image Analysis

ImageGear provides a histogram-generating function (and a related function to clear histogram bins) to assist you in image analysis. For an 8-bit image, whether 8i or 8-bit gray level, these functions can be called as follows:

```
HIGEAR hIGear; /* HIGEAR handle of image */
DWORD dwHistoBins[256]; /* Array of bins for counting */
IG_IP_histo_clear ( &dwHistoBins, 256 ); /* Clear the bins */
IG_IP_histo_tabulate (hIGear, &dwHistoBins, 256, NULL, 1, 0 );
/* Tabulate: */
```

In the above call, 256 `DWORD` bins are provided (one for each possible pixel value that can occur). The function will examine each pixel in the image bitmap, and will increment the bin corresponding to that value. (That is, the bin `dwHistoBins[pixel value]` will be incremented.) Upon return, you will have a count of the number of occurrences of each pixel value.

The fourth argument in the above call lets you specify the address of a rectangular region, if you want to restrict the tabulating to include only a portion of the image. When `NULL`, the whole image is included in the tabulation.

The fifth argument above can be set larger than **1**, if you want to increase the speed of the tabulation for a large image. **1** means that every raster line of pixels will be included in the tabulation. A larger value would result in raster lines being skipped.

The final argument in a call to **IG_IP_histo_tabulate()** is relevant for 24-bit images only. Its use is shown in the following example:

```
HIGEAR hIGear; /* Handle of a 24-bit image */
DWORD dwHistoBins[256]; /* Array of bins for counting */
IG_IP_histo_clear ( &dwHistoBins, 256 );/* Clear the bins */
IG_IP_histo_tabulate (hIGear, &dwHistoBins, 256, NULL, 1,
IG_COLOR_COMP_R ); /* Tabulate:   */
```

For 24-bit images, only 1 color channel is tabulated by a given call. That is, only 1 byte of each 3-byte pixel is examined. The final argument tells which byte (Blue, Green, or Red) should be tabulated.

You can also call **IG_IP_histo_tabulate()** for 1-bit and 4-bit images. In these cases, the number of DWORD histogram bins you need would be 2 and 16 respectively. In any event, always remember to call **IG_IP_histo_clear()** before calling IG_IP_histo_tabulate(), unless it is your intention to accumulate the count into the existing contents of your bins.

Special Effects

Special effects are a group of image processing functions that distort the image in some way, but still allows the image to be viewed, but with reduced clarity. ImageGear's special effects image processing functions (note that their names begin with IG_FX_ …) include:

IG_FX_blur()	Makes your image appear blurred. See also **IG_FX_motion()**.
IG_FX_chroma_key()	Blends two images. See the section entitled Blending and Combining Images later in this document.
IG_FX_diffuse()	Diffuses your image by locally shuffling the positions of pixels.
IG_FX_emboss()	Gives a 3-D chiseled-in-stone-like look to an image, similar to the appearance of the face on an engraved coin.
IG_FX_motion()	Makes your image appear as if it was moving when the image was captured. You can specify the direction of the apparent motion.
IG_FX_noise()	Adds noise to an image (by altering some pixel values). This tends to make the image look old or worn
IG_FX_pixelate()	Gives groups of pixels identical values, effectively reducing the resolution of the image. Can make it appear that you have zoomed in on a smaller image.
IG_FX_posterize()	Reduces the number of colors in the image by making groups of adjacent palette colors identical in the palette.
IG_FX_spotlight()	Creates a "spotlight" effect by darkening pixels outside the circular area you specify.
IG_FX_stitch()	Provides an effect similar to **IG_FX_emboss()** above, except the output more closely resembles a quilted stitch pattern.
IG_FX_texture()	Gives your image a textured look by applying changes to neighborhoods of pixels according to the pixel values in a small texturing image that you provide.
IG_FX_twist()	Makes your image appear as if it is being viewed through a shower curtain by rotating neighborhoods of pixels.

IG_FX_watermark()	Superimposes a watermark that you provide onto your image.

Each of the above functions is called similarly: with the HIGEAR handle of the image, a specification of where to apply the effect, and an additional argument or arguments as needed for you to specify the degree of the effect, direction, or other control. Refer to Core Component API Function Reference for additional details on each of these special effect functions.

Blending and Combining Images

A number of ImageGear's image processing functions combine data from two or more HIGEAR images, altering one with the result that it is a "blend" or other combination of the original images. These functions invariably take at least two HIGEAR handles as arguments. The functions in this group include:

IG_IP_blend_with_LUT()	Performs weighted blend of image 2 into image 1, with the weighting factor determined by looking up the pixel/color value in your LUT.
IG_FX_chroma_key()	This Special Effects function replaces image 1 by image 2 only where the image 1 pixels are within a specified hue range.
IG_IP_color_combine_ex()	Combines the pixel values from separate 8 bit images to form a single 24-bit color image.
IG_IP_color_separate()	The opposite of **IG_IP_color_combine_ex()**. Separates a 24-bit image's Red, Green, and Blue color values, producing separate 8-bit gray-scale images (each with their own higher handle).
IG_IP_blend_percent()	Blends together the pixel values (if gray-scale) or color values (if 24-bit) of the two images, according to a percent parameter that you specify. You can specify, for example, to combine 80 percent of image 1's value, with 20 percent of image 2's value. For a 24-bit image, you can specify that only one color channel (Red, Green, or Blue), or that all three color channels, are to be blended.

When you use **IG_IP_blend_with_LUT()**, you provide the address of a 256-byte Look Up Table (LUT). ImageGear looks up image 1's pixel value or color value in your LUT to determine the percentages to blend for that pixel.

A different kind of blend is provided by Special Effects function **IG_FX_chroma_key()**. This function searches for pixels in image 1 that are within a color hue range that you specify. Only pixels within that color hue range are replaced from image 2. Arguments are provided for you to control the smoothness of the transition where replacement occurs, and a threshold pixel darkness below which replacement will not occur. You might use this to remove a person's picture from a plain background and to replace the original background with a new one. TV stations often use this technique to overlay the weather reporter onto a background of a weather map.

IG_IP_color_combine_ex() and **IG_IP_color_separate()** let you, respectively, assemble a 24-bit image from separate 8-bit gray level images, or disassemble a 24-bit image into separate 8-bit gray level images. In both cases, you can specify whether the 24-bit combination is standard RGB, or whether or not the 24-bit value is to be interpreted in terms of a different color space scheme. You may specify IG_COLOR_SPACE_RGB, or IG_COLOR_SPACE_IHS, or others. The complete list of supported color space schemes is provided in the file accucnst.h.

Image Correction

ImageGear provides several functions whose specific purpose is to correct an image's appearance. The functions in this group include:

IG_IP_despeckle()	Removes noise from a 1-bit image, using a 3x3 median filter on the image.
IG_IP_median()	Uses a median filter to reduce noise in 8-bit gray level and 24-bit images.
IG_IP_deskew_auto()	De-skews a 1-bit image that was created by reading a text file. You can specify a threshold angle below which de-skewing should not be performed.
IG_IP_deskew_angle_find()	Will analyze a 1-bit image and report to you the angle at which it is skewed.

In general, you would use the image correction functions with images that have been obtained by scanning text documents or graphical images.

Image Maintenance

ImageGear provides a group of image processing functions that perform image maintenance operations. All of these functions change the image bitmap, in some way changing either its size or format of the pixel data stored within it. The functions in this group include:

IG_IP_crop()	Crops an image to a rectangle that you specify, discarding pixels that fall outside the rectangularly area.
IG_IP_resize()	This also changes the dimensions of the image bitmap, but scales the image to fit the new dimensions.
IG_IP_thumbnail_create()	This creates a new HIGEAR image for you, of the dimensions you specify, scaling your original image to fit the new dimensions. The original image is not altered or deleted.
IG_IP_swap_red_blue()	Exchanges the first and third bytes of each pixel in a 24-bit image's bitmap, changing the order from BGR to RGB, or vice versa.

IG_IP_encrypt()	Encrypts your image bitmap, using a password string that you specify. To successfully decrypt the image bitmap that password will be needed.
IG_IP_decrypt()	Decrypts an image bitmap, according to a password string that you specify. If the password is the same as used with **IG_IP_encrypt()** then the decrypting will be successful and the original image bitmap will be restored.

Pixel Access Operations

ImageGear is equipped with several functions that let you get and set the values of individual pixels, rows or columns of pixels, and rectangular groups of pixels. This family of functions is referred to as the "pixel access" functions.

All functions include the acronym "DIB" in their names. Every pixel access function is part of a "`_get()/_set()`" pair of functions. In other words, for each pixel access type, you can obtain the value(s) of a pixel or group of pixels, and set the value(s) of a pixel or group of pixels.

To obtain descriptions of each pixel access function and view additional sample code, refer to Core Component API Function Reference. It is important to note that although DIBs are stored upside-down so that (0,0) refers to the lower left-hand corner, ImageGear's pixel access functions consider (0,0) to be the upper left-hand corner of the bitmap data.

NOTE: Be sure to read the section entitled Using Pixel Access Functions with ImageGear.

The pixel access functions have two modes of operation: legacy (prior to 14.5) and new (14.5 and beyond). The default mode is legacy, in which these functions behave the same way they did before 14.5. So if you have existing code using pixel access functions, you shouldn't need to update it for 14.5. If you're considering using the pixel access functions for new code and you don't need to work directly with higher bit depths, uncommon color spaces, or alpha/extra channel data included with the main channel data, we recommend using the legacy mode. This mode shields you from the added complexity of these new formats.

If you want to use the new formats, or achieve maximum efficiency, you can use the new mode of pixel access to get more direct access to the new storage system. If you use the new mode, you must be aware of the following differences as compared to legacy mode:

- For RGB images, color channel order is RGB (in legacy mode, it's BGR)
- You can access higher bit depths (i.e. 36-bit, 48-bit RGB)
- You can access other color spaces (i.e. LAB, YUV)
- Alpha and extra channel data is included on a per-pixel basis. For example, if you have a 24-bit RGB image with an 8-bit alpha channel, the pixel data will look like RGBA, RGBA, RGBA and so on, where R, G, B, and A are each one

byte.

- Pixel packing and raster padding are as follows:

Packing Mode	Legacy	New
IG_PIXEL_PACKED	1, 4bpp packed, DWORD padding	1bpp packed, DWORD padding
IG_PIXEL_UNPACKED	No packing, BYTE padding	No packing, DWORD padding

Note that 1-bit channels are the only channels that are packed in the new mode - 8 pixels stored in each byte. All other channel depths are stored as 1, 2, or 4 byte values. A channel value with depth of 2-8 bits will be stored in one byte, 9-16 bits in two bytes, and 17-32 bits in four bytes. Channel values with depths of 33-64 bits are possible on 64-bit platforms and are stored in eight bytes.

To use the new mode, you need to set the `DIB.PIX_ACCESS_USE_LEGACY_MODE` global control parameter to `IG_PIX_ACCESS_MODE_NEW`. For example:

```
AT_MODE pixAccessMode = IG_PIX_ACCESS_MODE_NEW;
IG_gctrl_item_set("DIB.PIX_ACCESS_USE_LEGACY_MODE",
AM_TID_AT_MODE,
   &pixAccessMode, sizeof(pixAccessMode), NULL);
```

Allocating Space for ImageGear Pixel Access Functions

A common thread for all `_get()` functions is that you must provide an array with enough space to accommodate the data that you will receive. For 1, 4, and 8-bit images, one byte per pixel must be allocated. For a 24-bit image, you must allocate 3 bytes for each pixel. One exception to this rule is when getting a rectangular area of pixels. These ImageGear functions permit you to store the data packed, so that for a 1-bit image, you could store 8 pixels per byte, and so on.

Remember that the RGB data in a DIB is stored backwards so that the pixel values that you will receive will be in the order of Blue-Green-Red (unless you have changed the order of the image bitmap bytes using **`IG_IP_swap_red_blue()`**, or one of your own image processing functions).

NOTE: It is very important that you always allocate 3 bytes for each pixel when dealing with 24-bit images.

Getting and Setting Individual Pixels

There is one ImageGear function for getting and one ImageGear function for setting the value of an individual pixel: **`IG_DIB_pixel_get()`** and **`IG_DIB_pixel_set()`**, respectively. You must supply each of these functions with the `HIGEAR` handle of the image you are processing, and with an **x** and **y** position from which to get and then set the pixel's value. As with all pixel access functions, you should also provide an array to store the pixel value. For a 1, 4, or 8-bit image, one byte needs to be allocated, for a 24-bit image, allocate 3 bytes, for a

32-bit image, allocate 4 bytes, for a 9-16-bit gray level image, allocate 2 bytes. As noted in the previous section, the pixel access functions consider the coordinates 0,0 as the upper left-hand corner of the bitmap data.

Getting and Setting Linear Groups of Pixels

ImageGear lets you get and set the values of linear groups of pixels that run in a horizontal, vertical, or diagonal direction.

IG_DIB_column_get()	Gets the values of a variable-length (vertical) column of pixels
IG_DIB_column_set()	Sets the values of a column of pixels
IG_DIB_line_get()	Gets the values of a variable-length line of pixels
IG_DIB_line_set()	Set the values of a line of pixels
IG_DIB_raster_get()	Gets the values of a full raster line of pixels
IG_DIB_raster_set()	Sets the values of a full raster line of pixels
IG_DIB_row_get()	Gets the values of a variable-length (horizontal) row of pixels
IG_DIB_row_set()	Sets the values of a row of pixels

Raster and row access API allow packing more than one pixel per byte, you may set the `nFormat` argument to `IG_PIXEL_PACKED`. For more details, see Core Component API Function Reference.

If you are using a 24-bit image, you must allocate three bytes per pixel, whether the data is packed or unpacked. The data from a 24-bit image will be stored in BGR (blue-green-red) order. See the section entitled Allocating Space for ImageGear Pixel Access Functions for more details.

The **`IG_DIB_column_get()`** function requires that you set the `nX` parameter to the horizontal position of the column that you would like to get, and also the first and last row from which you would like to get the pixel values. For this you set the values of `nY1` and `nY2`. You also give it a pointer to and `nLenBytes` (size in bytes) of the buffer to which you will store the pixel values. This function will return to you the actual number of pixel values acquired: `lpNumPixels`. If the buffer size of `nLenBytes` is not large enough to accommodate the number of pixels specified by `nY1` and `nY2`, the line of pixels will be truncated, and you will not receive all of the pixels that you had specified.

`IG_DIB_column_set()` does just the opposite as its `_get()` counterpart. It takes the pixel values in the buffer and transfers a specified number of them to a `HIGEAR` image at row `nX`, and columns `nY1` through `nY2`. You also supply this function with the number of pixel values that you will be setting, where:

```
max # of pixels to set = (nY2 - nY1) + 1
```

`IG_DIB_row_get()` and **`IG_DIB_row_set()`** work exactly like the above-mentioned pair of functions, except that they get and set the values of a *horizontal* row of pixels.

`IG_DIB_raster_get()` and **`IG_DIB_raster_set()`** work similarly to the functions described above, except that they will get and set the values of a *full* (horizontal) raster line of pixels. These functions are quite a bit easier to use than the above

functions, however, because you do not need to supply the beginning and ending position of the line. Also, while you must allocate sufficient memory for the data, you do not need to tell ImageGear what number of bytes your buffer contains.

IG_DIB_line_get() and **IG_DIB_line_set()** give you access to the line of pixel values between any two sets of points in the image, i.e. a diagonal line, a horizontal line or a vertical line. For this reason, each requires that you set two **x** coordinates and two **y** coordinates. If you were to give equal values to either the **x** pair of coordinates or the **y** pair of coordinates, you would specify a line that was strictly horizontal or vertical, respectively. These two functions require that you give the size in bytes of the buffer and its address.

Getting and Setting a Rectangular Area of Pixels

The ImageGear functions for getting and setting the values of a rectangular area of an image are **IG_DIB_area_get()** and **IG_DIB_area_set()**, respectively. They are the same as the other pixel access functions insofar as they take a HIGEAR and a pointer to the address of the array allocated for storing the pixel values. Rather than directly taking 2 or more x and y coordinates, however, each of these functions takes an AT_RECT structure (defined in Gear.h), whose members specify the top, left, bottom and right of a rectangle within the image. Each function also takes an AT_MODE constant for the nPixelFormat parameter. This parameter may be set to one of the following values:

IG_DIB_AREA_DIB	Stores or transfers the pixel values; 8 pixels per byte for 1-bit images, 2 pixels per byte for 4-bit images, 1 pixel per byte for 8-bit images, and one pixel per 3 bytes for 24-bit images. Please also note that each row of the data will be padded to DWORD edge.
IG_DIB_AREA_UNPACKED	Stores/transfers unpacked pixel values at 1 pixel per byte for 1-, 4-, and 8-bit images, and one pixel per 3 bytes for 24-bit images.

The constant IG_DIB_AREA_DIB stores data the way that it is stored in an uncompressed DIB format, with the pixels packed so that each row is padded to a multiple of 4 bytes in length. The pixels will be stored left justified - most significant bit first. IG_DIB_AREA_UNPACKED will store the pixels right justified - least significant byte first.

NOTE: ImageGear's pixel access functions consider the coordinates 0,0 to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers 0,0 to refer to the lower left-hand corner of the bitmap.

Using Pixel Access Functions with ImageGear

If you are using any of the pixel access functions to *get* pixel values, you may get pixel data from an image of any bit depth (1, 4, 8, 9-16-bit gray level, 24, or 32). However, if you wish to *set* pixel values of a 1-bit image using, there are two peculiarities to consider:

The following pixel access functions work with the standard DIB format only:

IG_DIB_area_set()

IG_DIB_column_set()

IG_DIB_line_set()

IG_DIB_pixel_set()

IG_DIB_row_set()

If you did not first convert the runs-compressed image to a standard DIB you will receive this error: IGE_OPERATION_IS_NOT_ALLOWED.

TIP: To be thorough, you should make a check of the 1-bit image to make sure that it is actually compressed. To do so call **IG_image_compression_type_get()** before converting the image from compressed DIB to a standard DIB. A return value of IG_BI_RLE means that the image is compressed.

The **IG_DIB_raster_set()** function needs special attention when used with ImageGear. If you call IG_DIB_raster_set() for all of the rasters in the image, you do not have to convert the image to a standard DIB first.

Clipboard operations

The “clipboard” family of functions consists of a handful of straightforward functions that help you to meet all of your clipboard cut and paste needs very easily. With this function group, you can cut or copy all or a portion of an image to the system clipboard, paste the contents of the clipboard into a new HIGEAR image, or even “paste-merge” the contents of the system clipboard into a pre-existing image. You can also check for the existence of data in the clipboard, and check the size of an image in the clipboard. For separate descriptions of each clipboard function and additional sample code, please refer to Core Component API Function Reference.

Copying/Cutting to the Clipboard

You may cut or copy the entire HIGEAR image, or just a specified rectangular portion of the image, to the clipboard. To copy to the clipboard simply call the function **IG_clipboard_copy()**with the image’s HIGEAR handle, and the coordinates of the AT_RECT rectangle that you would like to save to the clipboard. Pass NULL as the rectangle’s value to if you want to copy the entire image to the clipboard.

To cut to the clipboard, call **IG_clipboard_cut()**. The only difference in the prototype of these functions is that IG_clipboard_cut() contains an extra argument for specifying what color pixel to use to replace the pixels that are “cut away”. This pixel color argument is usually set to black or white.

Checking the Contents of the Clipboard

ImageGear provides two functions for examining the contents of the clipboard: **IG_clipboard_paste_available_ex()** and **IG_clipboard_dimensions()**. The first of these functions will let you know whether there is an image in the system clipboard. It is recommended that you always call this function before pasting from the clipboard, and also before calling IG_clipboard_dimensions().

IG_clipboard_paste_available() returns a AT_BOOL value, where TRUE means that there is a paste-able image in the clipboard.

IG_clipboard_dimensions() returns three values to you: the width and height of the image (in pixels), and the number of bits per pixel of the image on the clipboard. Using these values, you can determine whether or not the image dimensions are appropriate for your purposes.

Pasting an Image from the Clipboard

There are two ImageGear functions for pasting the image from the clipboard: **IG_clipboard_paste()** and **IG_clipboard_paste_merge_ex()**. The first of these two functions will create a new HIGEAR image and paste in the contents of the clipboard. IG_clipboard_paste_merge() will paste the clipboard image into an existing HIGEAR image at the position that you specify. If the clipboard image's width is greater than the image being pasted to, it will automatically be cropped to fit; the size of the original HIGEAR image will not change.

Before you call IG_clipboard_paste_merge(), you can call the function **IG_clipboard_paste_op_set()** to set the kind of arithmetic operation to apply to the pixels of the two bitmaps that intersect during the paste-merge. IG_clipboard_paste_op_set() takes an AT_MODE constant (defined in accucnst.h) that has a prefix of IG_ARITH_. The full group of arithmetic constants is listed under the function description for IG_clipboard_paste_op_set() in Core Component API Function Reference. ImageGear also supplies a companion reading function **IG_clipboard_paste_op_get()** to read the current setting for the paste-merge arithmetic operation. See Example code below:

```
DIMENSION nWi, nHi;
UINT nBpp;
BOOL bPasteAvail;
AT_ERRCOUNT nErrcount;
HIGEAR hIGear, hIGear2;
AT_RECT rcClipRect;
nErrcount = IG_load_file(“picture.bmp”, &hIGear);
if (nErrcount == 0)
{
   nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi,
&nBpp );*/
   if ( nErrcount == 0 ) /* If valid image dimensions */
   {/* send the bottom half of the image*/
      rcClipRect.top = nHi/2; /* to the clipboard */
      rcClipRect.left = 0;
      rcClipRect.right = nWi - 1;
      rcClipRect.bottom = nHi - 1;
      nErrcount = IG_clipboard_copy (hIGear, &rcClipRect);
   }
if (nErrcount == 0)
{
/*load a second image into which to merge the clipboard
contents*/
      nErrcount = IG_load_file(“picture2.bmp”, &hIGear2);
}
if (nErrcount == 0)
{
```

```
        nErrcount = IG_clipboard_paste_available(&bPasteAvail);
        if (bPasteAvail == TRUE)
        {
        /* set the paste-merge arithmetic operation to Img1^Img2 */
        nErrcount = IG_clipboard_paste_op_set(hIGear,
                  IG_ARITH_XOR);
        /* merge clipboard's rectangular contents with upper left
         corner at position 0,0 */
           nErrcount = IG_clipboard_paste_merge(hIGear2, 0 , 0);
           }
    }
```

Printing Images

To print to a Postscript printer, save the image as an encapsulated Postscript and copy the file to the printer. You may want to open a connection with the printer directly and save the image with **`IG_save_FD()`**.

Thread Safety

ImageGear and its associated components are 100 percent thread-safe. The implementation of thread safety in ImageGear maximizes the performance of threaded applications on multi-CPU computers, providing the benefit of performance scalability at a huge cost savings over other implementations. Every ImageGear API function can be executed within a thread.

Five thread safety APIs are explained in detail in "Core Component API Function Reference".

- **`IG_thread_data_ID_associate()`**
- **`IG_thread_data_ID_get()`**
- **`IG_thread_local_data_cleanup()`**
- **`IG_thread_image_lock()`**
- **`IG_thread_image_unlock()`**

In most cases no additional API calls are required to achieve thread safety.

`IG_thread_data_ID_associate()`, `IG_thread_data_ID_get()`, and `IG_thread_local_data_cleanup()` functions only provide additional flexibility allowing thread customized ImageGear settings.

`IG_thread_image_lock()/IG_thread_image_unlock()` are only required in the situation where several threads are accessing the same `HIGEAR` concurrently AND at least one of these threads performs an operation that modifies/deletes `HIGEAR`.

NOTE: The global group of ART settings cannot be used in multi-threaded application. You have to use this function to associate a new local group of settings for each new thread. For more detailed information please see the example code for **`IG_thread_data_ID_associate()`** function.

NOTE: The access to the same PDF document from multiple threads is not permitted because the multiple threads cannot share Adobe PDF Library datatypes. PDF doc created/opened in the main thread can be only used from the main thread.

Working with Format Filters

ImageGear introduces significant improvements in the area of working with images stored on disk and in memory. This includes new additional API, improved scheme of setting image control parameters and more GUI controls to simplify and illustrate the process of working with file formats.

Every image format supported by ImageGear has its own filter, including the set of internal functions organized into the special structure that completely defines the filter's functionality. Every filter has its own banch of control parameters that can determine the manner in which it loads, saves and operates with images. Besides this, the control parameters used for the filters API are not constants (`IG_CONTROL_TIF_BUFFER_SIZE`) as they were in previous versions of ImageGear, but a string (`"BUFFER_SIZE"`). Please see detailed information about the control parameters for each format filter in the ImageGear Supported File Formats Reference section.

Every format filter has its own list of supported features determined by `IG_FLTR_` flags. The table below describes the meaning of each of these flags:

`IG_FLTR_DETECTSUPPORT`	Means that filter supports autodetection.
`IG_FLTR_PAGEREADSUPPORT`	Means that filter supports single page file reading.
`IG_FLTR_MPAGEREADPSUPPORT`	Means that filter supports multi-page file reading.
`IG_FLTR_PAGEINSERTSUPPORT`	Means that filter supports single-page file writing.
`IG_FLTR_MPAGEWRITEPSUPPORT`	Means that filter supports multipage file writing.
`IG_FLTR_PAGEDELETESUPPORT`	Means that filter supports page deleting from multi-page file.
`IG_FLTR_PAGESWAPSUPPORT`	Means that filter supports page swaping in multi-page files.
`IG_FLTR_MPDATASUPPORT`	Means that filter supports faster multi-page access by storing private format data (used only with `IG_mpi_...` and `IG_mpf_...` API).

Please see information about the supported features for each format filter in the ImageGear Supported File Formats Reference section.

ImageGear supports different types of image compression. Currently every ImageGear format filter has its own list of supported compressions, determined by `IG_COMPRESSION_` constants for different types of images. For information about the supported compressions for every format filter, see the ImageGear Supported File Formats Reference and ImageGear Supported Compressions Reference section.

Using the Format Filters API for Image Loading and Getting Information.

The ImageGear Format Filters API allows you to load an image into memory, detect the format of an image file, get info about the features supported by ImageGear for this format, provide image control parameters, work with pages in multi-page files and save an image in a single- or multi-page file.

In the case of loading and saving image files, you can choose which functionality group (`IG_fltr_ ...()` or `IG_load/save_ ...()`) is more useful for your application.

To import an image from a file format to an internal memory representation, you can use the function:

```
IG_fltr_load_file(const LPSTR lpszFileName, UINT nPageNumber,
LPHIGEAR lphIGear);
```

If the image file is on a mass storage device, such as a disk, then the first parameter specifies its file name. The second parameter determines the page number (starting from 1) if the image file contains more than 1 page. The last parameter is a handle of internal memory representation. This received handle can be used then to perform different operations on the image, such as image processing or export operation. The file specified by `lpszFileName` should be in one of the ImageGear supported image file formats; ImageGear detects this format automatically. If the format is invalid or unsupported, this function puts an error into the error stack. This function is similar to **`IG_load_file()`**, but the last one always imports the first page.

TIP: Actually you can use other ImageGear functions for loading images from disk and from memory, such as `IG_load_...()` functions. They are very convenient when you don't need to work with a large amount of images and you do not need to optimize the speed of loading multiple pages from file. But if you need to work with a large range of image files, especially multi-page files, and need your application to work faster, choosing images with determined parameters such as the type of file format, compression, page number, size, supported features, control parameters, we recommend that you use the `IG_fltr_...()` functions.

Detecting the File Format

You can detect the format of image file before and after loading it in `HIGEAR` handle using functions `IG_fltr_detect_...()`:

```
IG_fltr_detect_FD(INT fd, LONG  lOffset, LPATE_MODE lFileType)
IG_fltr_detect_file(const LPSTR lpszFileName, LPATE_MODE
lFileType)
IG_fltr_detect_mem(void FAR lpImage, DWORD dwSize, LPATE_MODE
lFileType)
```

The first function detects the format by scanning its File Descriptor `fd` starting from the `lOffset` position in the file. The second `IG_fltr_detect_file()` function determines the format of image located in the file by given filename. Function `IG_fltr_detect_mem()` determines the format of image located in memory buffer. All three functions return the type of image format as `IG_FORMAT_` constant, determined in `accucnst.h` file. The code fragment below allows you to sort image files by their formats and then choose how to use them:

```
AT_ERRCOUNT nErrCount = IGE_SUCCESS;/* will hold returned error
count */
AT_MODE nFormatID;
…
nErrCount = IG_fltr_detect_file( "image.tiff", &nFormatID );
if( nFormatID==IG_FORMAT_TIF )
{
…
}
```

For better performance or for some other purposes, use the special functions **IG_fltr_load_file_format()** and **IG_fltr_pagecount_file_format()** if the image format is known. Those functions accept the first parameter as the format ID and do not perform usual detect operations like **IG_fltr_load_file()** does, but immediately start to operate with the data, assuming it from the specified format. If the data is invalid or the image format is different, then an error is placed in the error stack.

Getting Information and Sorting Images

To get information about every ImageGear format filter use **IG_fltr_info_get()** function. It returns information about the features supported by ImageGear for this file format (as `IG_FLTR_` flags), the short and full names of the filter's file format and the names of the default file's extension:

```
nErrCount = [lpfn]IG_fltr_info_get( nFormat, &dwFlags,
szShortName, sizeof(szShortName), szFullName,
sizeof(szFullName), szDefExt, sizeof(szDefExt) );
```

Together with this function, you can use the **IG_fltr_formatlist_sort()** function to sort file formats in alphabetic order based on the short name returned by the `IG_fltr_info_get()` function through its third `lpShortName` parameter.

If you want to determine the list of filters that support the `IG_FLTR_` features you can use the function:

```
IG_fltr_formatlist_get(DWORD dwFlags, LPAT_MODE lpFormatList,
UINT nFListSize, LPUINT lpnFListCount)
```

Through the `dwFlags` argument, you can specify any combination of features (for instance, `IG_FLTR_DETECTSUPPORT | IG_FLTR_PAGEREADSUPPORT | IG_FLTR_MPAGEREADPSUPPORT`), and a list of filters that support those features will be returned through the second parameter (`lpFormatList`). See the description of this function in the Core Component API Function Reference for detailed information about the parameters. If you want to determine the value of the `nFListSize` argument, first set `lpFormatList` to `NULL`. You can also use this function together with **IG_fltr_formatlist_sort()** to get the list of filters that support necessary features and sort them by filter name:

```
UINT              nFilterSize;
UINT              nFListSize;
LPAT_MODE         FList = NULL;
LPCHAR            lpFilter = NULL;
LPSTR     str;

IG_fltr_formatlist_get( IG_FLTR_PAGEREADSUPPORT, NULL, 0,
&nFListSize );
```

```
FList = malloc( nFListSize*sizeof(AT_MODE) );
if( FList!=NULL )
{
IG_fltr_formatlist_get( IG_FLTR_PAGEREADSUPPORT, FList,
nFListSize, NULL );
IG_fltr_formatlist_sort( FList, nFListSize );
...
}
```

You can get information about any page of a multi-page file without loading it into memory if you use the function **IG_fltr_pageinfo_get()**. Return information consists of the name of the file format (as IG_FORMAT_ constant), the type of image compression (as IG_COMPRESSION_ constant), and such info as bpp, width and height of the page-image.

Another two IG_fltr_...() functions allow you to work with pages in a multi-page file without loading it in memory:

```
IG_fltr_pageswap_file (const LPSTR   lpszFileName, AT_MODE
nFormatType,  UINT  Page1, UINT  Page2)
IG_fltr_pagedelete_file (const LPSTR   lpszFileName, AT_MODE
nFormatType,  UINT  nStartPage, UINT  nRange)
```

Both functions accept the filename of the multi-page image, format filter ID and two integer parameters as page numbers (assuming numeration from 1). The first function swaps pages in a multi-page image specified by page numbers, but the second function deletes pages from it.

Before using these functions you have to determine whether or not ImageGear supports the corresponding features for the necessary file format. This can be done using the function **IG_fltr_info_get()** and inspecting flags returned through the second parameter for the specified file formats. If this value contains the flag IG_FLTR_PAGESWAPSUPPORT, then the format filter does support the IG_fltr_pageswap_file() operation. If it has also the flag IG_FLTR_PAGEDELETESUPPORT, then it supports the IG_fltr_pagedelete_file() operation**.**

NOTE: **SWAP** and **DELETE** operations may or may not physically reorder multi-page files. For example, the page delete operation may not really reduce the size of the file – just update the file format structures and remove the specified pages from it.

Using Format Filters API for Image Saving

The ImageGear filters API provides three saving functions that make your application much faster and more flexible when working with image saving:

```
IG_fltr_save_file (HIGEAR hIGear, const LPSTR lpszFileName,
AT_LMODE lFormatType, UINT nPageNumber, AT_BOOL bOverwrite)
IG_fltr_compressionlist_get (LPAT_DIB lpDIB, AT_MODE nFormatID,
LPAT_MODE lpComprList, UINT nCListSize, LPUINT lpnCListCount)
IG_fltr_savelist_get (LPAT_DIB lpDIB, LPAT_MODE lpnFilterList,
UINT nFListSize, LPAT_LMODE lpSaveList, UINT nSListSize, LPUINT
lpnSListCount)
```

The function IG_fltr_compressionlist_get() allows you to get information about all compressions (as IG_COMPRESSION_ constants returned in lpComprList list) that are available when saving the image to the file format specified by the nFormatID (as IG_FORMAT_ constant) parameters. The first parameter allows you to specify information about the image to be saved. If this parameter is NULL then the function returns all available compressions, otherwise it returns compressions that are applicable to a given image. For example, G3/4 compressions are only applicable for bi-tonal images, but JPEG compression is only applicable for color images with 8 bits or more per pixel.

Function **IG_fltr_savelist_get()** is more advanced and allows you to prepare the list of applicable compressions not for the single format but for the list of formats. It provides the ability to quickly build the list of applicable parameters that can be used, for instance, for the third (lFormatType) parameter of **IG_fltr_save_file()**. The main returned value lpSaveList contains a list of combined values (nFormat | (nCompression<<16)) = (IG_FORMAT_ | IG_COMPRESSION_<<16) available for a given image to save. You can get values of nFormat from lpnFilterList provided through the second parameter. For instance, if you specified lpnFilterList as IG_FORMAT_TIF, then the returned lpSaveList will be looked at as a list of (IG_FORMAT_TIF | IG_COMPRESSION_) values where IG_COMPRESSION_ is a list of all the compressions available for the TIFF image with the current lpDIB.

If lpnFilterList is NULL then this function uses all available formats registered in ImageGear and returns the list with all currently available formats and compressions.

The basic saving function of the ImageGear Filters API **IG_fltr_save_file()** works similarly to the **IG_save_file()** function, but has additional arguments that enhance your work when saving the image in a multi-page file. The nPageNumber argument allows you to specify the number of the page in an already existing multi-page file where you want the saved page to be placed. The last argument bOverwrite allows you to determine the mode of how to work with multi-page image files. The TRUE value completely overwrites the file and places a single page there, but a FALSE value means that the existing file will be expanded with one additional page specified by the nPageNumber parameter.

You can process the **IG_fltr_compressionlist_get()** and/or **IG_fltr_savelist_get()**functions before using **IG_fltr_save_file()**to determine the file format and compression type you specified in the lFormatType argument (as IG_SAVE_ constant, see accucnst.h file).

Using Format Filters API for Filter Control

Almost every format filter in ImageGear has some attributes that it depends on while processing its operations such as **READ**, **WRITE** and etc. Those attributes may be attributes declared by the format filter specification or it may be specific to its implementation by ImageGear. Anyway, ImageGear has a general public interface implemented and named as **format filter control parameters**. Every such control parameter is identified by the format filter and string name and has an associated type of acceptable value, the value itself and the default value. The sense of each control parameter is filter specific. The ImageGear Supported File Formats Reference section in Chapter File Format Reference, where descriptions of filters are located, also contains a description of each control parameter for each format filter.

The ImageGear Filters API has three new powerful functions that allow you to get/set and retrieve info about every supported filter control parameter:

```
IG_fltr_ctrl_list(DWORD  dwFormatID, LPUINT lpnCount, LPDWORD
lpArray, DWORD dwArraySizeInBytes)
IG_fltr_ctrl_get(DWORD dwFormatID, const LPCHAR lpcsCtrlName,
AT_BOOL    bGetDefault, LPAT_MODE lpnValueType, LPDWORD
lpdwValueSize, LPVOID       lpBuffer, DWORD dwBufferSize)
IG_fltr_ctrl_set(DWORD dwFormatID, const LPCHAR lpcsCtrlName,
LPVOID lpValue, DWORD dwValueSize)
```

The first function `IG_fltr_ctrl_list()` allows the application to get the list of names of all control parameters supported by the format filter identified by `dwFormatID`. The Application is responsible for allocating the buffer `lpArray`, but ImageGear sets the elements of this array as pointers to strings with control parameter names.

You can use these control parameter names as input values for the second argument for the next two functions - `IG_fltr_ctrl_get()` and `IG_fltr_ctrl_set()`. The first function is used to get the value of a given control parameter of a given format filter, and it may return either its current value or its default value - that is controlled by the `bGetDefault` argument. The `TRUE` value returns default value, but `FALSE` returns the current value. The value itself is copied into a buffer that the application provides through the `dwBufferSize` argument.

The third function `IG_fltr_ctrl_set()`allows you to set a new value for the control parameter. The last two arguments of this function specify the data to be set, and ImageGear always treats this data as a type that can be gotten by the `_get()` function.

NOTE: If the actual size of the new value is less than 4 bytes, then `lpValue` is treated as the value itself, otherwise it is treated as a pointer to the value.

The following code fragments demonstrate how to use these functions:

This example demonstrates how to get the names of all supported control parameters for the TIFF format filter:

```
/* getting the total number of parameters */
nErrCount = IG_fltr_ctrl_list(IG_FORMAT_TIF, &nCount, NULL, 0);
if(!nErrCount && nCount > 0)
{
/* allocate required buffer to keep all names */
lpOptList = malloc(nCount * sizeof(DWORD));
   if(lpArray)
   {
   nErrCount = IG_fltr_ctrl_list(IG_FORMAT_TIF, NULL,
lpOptList, nCount * sizeof(DWORD));
```

NOTE: Please remember that filter control parameters you work with using `IG_fltr_ctrl_...()` functions are now strings. You can see them in the **"Filter Control Parameters"** Tables for every file format in the ImageGear Supported File Formats Reference section.

This example demonstrates how to get and set the value of the TIFF control parameter named “BIG_ENDIAN“ :

```
char DocumentName[_MAX_PATH];
AT_BOOL bDefBigEndian, bOldBigEndian;
...
/* get current value of BIG_ENDIAN control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "BIG_ENDIAN", FALSE, NULL,
NULL, (LPVOID)&bOldBigEndian, sizeof(hOldBigEndian));

/* get default value of BIG_ENDIAN control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "BIG_ENDIAN", TRUE, NULL, NULL,
(LPVOID)&bDefBigEndian, sizeof(hDefBigEndian));

/* get current value of DOCUMENT_NAME control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "DOCUMENT_NAME", FALSE, NULL,
NULL, DocumentName, sizeof(DocumentName));

/* set new value to BIG_ENDIAN control parameter */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "BIG_ENDIAN", (LPVOID)TRUE,
sizeof(AT_BOOL));

/* set new value to DOCUMENT_NAME control parameter */
strcpy( DocumentName, “This is a test string for DocumentName“
);
IG_fltr_ctrl_set(IG_FORMAT_TIF, "DOCUMENT_NAME",
(LPVOID)DocumentName, sizeof(DocumentName));
```

NOTE: In the case of TXT Filter to set the `LINES_PER_PAGE` and `CHAR_PER_LINE` control parameters, do not set the `POINT_SIZE` control parameter because setting the `PAGE_WIDTH`, `PAGE_HEIGHT` and `POINT_SIZE` parameters provides sufficient page description, and `LINES_PER_PAGE`, `CHAR_PER_LINE` options are ignored. To set these parameters you should set `POINT_SIZE` option to zero.

Non-Image Data Processing

Such complex format filters as EXIF-JPEG, EXIF-TIFF, TIFF, JPEG, PNG and some others contain a lot of non-image data generally refered to as meta-data. ImageGear provides a machanism for reading the meta-data during image loading and modifying it during image saving. Non-image data itself can be of any possible complex type and completely depends on the nature of the file format. ImageGear processes this complex data through a single interface and allows uniform processing that does not depend on the actual data format and starts from information fields of such simple formats as BMP and PCX, up to the complex metadata support in the EXIF filter and the IPTC non-image data format in such filters as TIFF and JPEG.

NOTE: While EXIF-JPEG, EXIF-TIFF is a separate image file formats using JPEG or TIFF image data compressions, IPTC is a format used only for non-image data storage in such imaging format filters as JPEG, TIFF. For more detailed information about these formats, see the EXIF-TIFF Non-Image Data Structure, EXIF-JPEG Non-image Data Structure and the IPTC Non-Image Data Structure sections in Appendix Appendix B: Non-Image Data Storage as well as the EXIF-JPEG and EXIF-TIFF sections in the File Format Reference Chapter.

ImageGear is responsible for translating this format-dependent data into a standard uniform format. There are at least two operations that include such data processing:

- **image loading** - during image loading some additional data needs to be loaded and uncompressed into the set of values of standard types for further processing.
- **image saving** - during image saving there should be a way to change already defined values and add new values.

The next section defines the uniform data format that is used for such data exchange.

Non-Image data format

The key thing of non-image data processing in ImageGear is a uniform data format that is used to convert to and from the format filter. As soon as the format filter decodes the data fields one after another during the loading operation, and encodes it in the reverse direction during the saving operation, all data consists of the set of items where each item is a minimal atom of information. The order of items is fixed, and the format filter processes item after item in the given order. The same order is used when data is passed through the stream.

Let's look at the low-level format of the data. It consists of the list of items where each item represents a minimal unit of information. Each item also should have some unique name that allows you to connect it with the physical value inside of the file format. The definition of the data item can be described by the following fields:

```
typedef struct tagAT_DATALIST_ITEM{
   AT_MODEFormatID
   LPCHAR Name;
   DWORD  Id;
   AT_MODE Type;
   LPVOID Value;
   AT_MODE ValueType;
   DWORD  Length;
   AT_MODE ValueAccessMode;
   }AT_DATALIST_ITEM;
```

Please see the descriptions of these fields in the Table below:

FormatID	The ID of the filter that reads or writes a file (IG_FORMAT_… constant value)
Name	The name of the item. Can be any string value.
Id	Numerical ID of item. Can be any value of DWORD size.
Type	Specifies the type of item and reflects the status of the given record. Possible values are: ■ IG_METAD_VALUE_ITEM – this value specifies that the current item is a value of the simplest type and the field Value contains the actual value of the item, and ValueType contains the identifier of the type of this item. ReadOnly can be either TRUE (read-only) or FALSE (read/ write). Name and/or Id contains textual and numerical identification of the item. ■ IG_METAD_LEVEL_START – this value specifies that the current item opens the sublevel of items and all the next items up to the corresponding item with the LEVEL_END value belonging to this sublevel. ■ IG_METAD_LEVEL_END – this value closes the current sublevel and tells that next item belongs to a higher level.
Value	Contains the value of the item in the case of Type = IG_METAD_VALUE_ITEM. **NOTE:** Possible values of this field are fixed and define exact list of allowed data types. It also depends on ImageGear platform and FLTR.METADATA_FORMAT global control parameter. This global parameter has two allowed values: “text” and “binary”. See the section Metadata Structure “ValueType” and “Value” for the possible values.
ValueType	Contains the type identifier of the item in the case of Type = IG_METAD_VALUE_ITEM. Possible values of this field are fixed and define the exact list of allowed data types. See the section Metadata Structure “ValueType” and “Value” for possible values.
Length	Identifies the number of values to be written. For AM_TID_TXT_STRING it should indicate the number of characters in the string excluding last null character (basically the length of the string). For AM_TID_RAW_DATA it should indicate the number of bytes that the raw data occupies. For the rest of the types it should indicate the number of values of the type, which textual representation is encoded into “Value”.

ValueAccessMode	Identifies wether data can be changed or not. “Read only” value means that its value is information only and cannot be changed after setting the initial value. It also means that its value will be ignored during a WRITE operation.

So, this data structure allows to “linearize” hierarchical and complex data into the array of simplest data types.

You can transfer different non-image data using the general data structure described in this section. Please see Appendix Appendix B: Non-Image Data Storage.

Processing of non-image data through filter callback functions

There are two working scenarios of how ImageGear processes non-image data.

The first one is the **LOAD** operation:

1. Application registers special callback function of type **`LPAFT_IG_METAD_ITEM_GET_CB`** .
2. The application calls some of the filter loading functions (like **`IG_fltr_load_file()`**), and during the **LOAD** operation, the format filter calls the registered callback function to pass data for each item decoded from the image.

The reverse **WRITE** operation is more complex:

1. Application registers callback functions of types **`LPAFT_IG_METAD_ITEM_SET_CB`** and **`LPAFT_IG_METAD_ITEM_ADD_CB`**.
2. Application calls some filter writing functions (like **`IG_fltr_save_file()`**).
3. While performing **WRITE** operation ImageGear uses callback functions to modify existing items or add additional items to required dataset.

NOTE: ImageGear provides new special `LPAFT_` callback functions for the non-image data processing described in this section. It also preserves the “old” callback functionality (`LPFNIG_` callback functions) required for the image processing control and perfection. Please see Working with ImageGear Callback Functions for detailed information about the structure of the ImageGear callback functionality.

You can see from the declaration below that `LPAFT_IG_METAD_ITEM_GET_CB` accepts parameters that provide all necessary information about one data item. All parameters except the first one are fields of the data structure `AT_DATALIST_ITEM` described in previous Non-Image data format section:

```
LPAFT_IG_METAD_ITEM_GET_CB(LPVOID lpPrivate, LPCHAR ItemName,
DWORD ItemID, AT_MODE ItemType, LPVOID ItemValue, AT_MODE
ValueType, DWORD ValueLength, AT_BOOL ReadOnlyValue )
```

By implementing and providing a callback function of this type, the application can receive every decoded item and process it as needed.

Some items from the dataset are informational only and cannot be changed during the **WRITE** operation. So, if `ReadOnlyValue` field is set to `TRUE` then item is not be changed during the **WRITE** operation. Actually, during this operation the format filter prepares all necessary items and puts required values to them to make sure that the file format itself is not violated. For instance, there can be the item that keeps the number of strips in the image, and if the value of this item is changed, the image cannot be loaded.

The format filter prepares a minimal set of items and default values for them, and before writing its values to the output stream it calls the callback function of type `LPAFT_IG_METAD_ITEM_SET_CB` so that the application can change its values. In addition, the format filter may call the function of type `LPAFT_IG_METAD_ITEM_ADD_CB` to get additional items to append the dataset. The application should provide its implementation in such a way that it returns `TRUE` until it is necessary to insert more items. If this function returns `FALSE`**,** all custom items have been added, and the filter can proceed

---

NOTE: It may happen that the application provides the item with the name or ID of a different type than the format filter is expecting. For example, the format filter may expect the item named **SOFTWARE** with text as a String, but the application provides its value as an Integer. In this case the filter may ignore this item and put the warning that this type of item is not expected. Also, the application may provide an item that the format filter is not able to handle because it does not fit the format defined by the corresponding image file format. In this case it may simply ignore the item and give a warning.

The exact specification of the callback function types can be found in the "Core Component API Function Reference" Chapter, but the following is a quick reference for better understanding:

```
LPAFT_IG_METAD_ITEM_SET_CB)(LPVOID lpPrivate, LPCHAR ItemName,
DWORD ItemID, AT_MODE ItemType, LPVOID ItemValue, AT_MODE
ValueType, DWORD ValueLength, AT_BOOL ReadOnlyValue, LPVOID
*NewItemValue, LPAT_MODE *NewValueType, LPDWORD *NewValueLength
)
```

`*NewItemValue`, `*NewValueType`, `*NewValueLength` are parameters of new value for given item. If `ReadOnlyValue` is `TRUE`, the value of this item is unchangable.

You can add a new non-image item during the filter **WRITE** operation using the callback function prototype:

```
LPAFT_IG_METAD_ITEM_ADD_CB(LPVOID lpPrivate, LPCHAR ItemName,
DWORD ItemID, AT_MODE ItemType, LPVOID ItemValue, AT_MODE
ValueType, DWORD ValueLength, AT_BOOL ReadOnlyValue )
```

All arguments in this function are parameters for the new item and its value.

To exchange the non-image tag information, provided by these three callback functions, between the application and the internal ImageGear structure levels you should use two functions:

```
IG_fltr_metad_callback_get(LPVOID *lpPrivate,
LPAFT_IG_METAD_ITEM_SET_CB    *lplpfnSetCB,
LPAFT_IG_METAD_ITEM_ADD_CB  *lplpfnAddCB,
LPAFT_IG_METAD_ITEM_GET_CB *lplpfnGetCB )
IG_fltr_metad_callback_set(LPVOID *lpPrivate,
LPAFT_IG_METAD_ITEM_SET_CB    *lplpfnSetCB,
LPAFT_IG_METAD_ITEM_ADD_CB  *lplpfnAddCB,
LPAFT_IG_METAD_ITEM_GET_CB *lplpfnGetCB )
```

The first function allows you to provide the current callback non-image tag data from the internal ImageGear structure to an application level during load/save processes. If some callback information is not necessary, you can set the respective argument of this function to `NULL`. For instance, if you do not need information about newly added non-image items, set `lplpfnAddCB = NULL`.

The second `_set()` function provides the new non-image callback data from the application level to the internal ImageGear level during the load/save processes. Again, if some callback information is not necessary you can set the respective argument of this function to `NULL`.

An example, how to use this callback functions is too complex to include into this manual. The GUI implementation that is provided in the source form demonstrates all aspects of working with these callback functions.

Updating non-image data in a file without loading and saving the image

Callback functions can be used to get and set non-image data during normal load and save operations. It is also possible to use callback functions with the **`IG_fltr_metad_update_file()`** function to operate on only the non-image data in a file. This function creates a new file with an exact copy of the source file's pixel data and with new non-image data. Pixel data is not decoded but copied directly from source to destination file. This function is currently supported with the following file formats only:

- TIFF (except for TIFF-JPEG)
- JPEG

`IG_fltr_metad_update_file()` obtains new non-image data from the following callback functions:

`LPAFT_IG_METAD_ITEM_SET_CB`, **`LPAFT_IG_METAD_ITEM_ADD_CB`**.

Using metadata update function

`IG_fltr_metad_update_file()` function can be used as follows:

- Load necessary page to get metadata;
- Change metadata (add / delete / change metadata tags or metadata values);
- Call `IG_fltr_metad_update_file()` function. `nPageNumber` and `lFormatType` parameter values should correspond to the loaded page and source file format.

The destination file will be a copy of the source file with the new non-image data for the specified page.

The application may then delete the source file and rename the destination file with the name of the source file.

Example of using IG_fltr_metad_update_file function

Usage of metadata update function is demonstrated in the Filter sample. Use the following steps to test this feature:

1. Open a TIF image using File/Open menu item;
2. Add or modify tag(s) using Image/Metadata/Data Structure dialog
3. Create a file with updated metadata using File/Update metadata menu item. Source file name and page number correspond to the last loaded page in the filter sample.

NOTE: All changes made in the Metadata dialog will be lost if any function that returns metadata is called before calling **`IG_fltr_metad_update_file()`** (loading a new file, getting file info or preview).

Working with ImageGear Color Profile Manager

ImageGear supports color profiles while performs color conversion operations. Detailed information about color processing based on profiles can be found at **www.color.org**. The following sections "Color Profile Basic Concepts", "ImageGear Color Profile Groups" and "Color Profile Manager API" describe how color profiles can be used with ImageGear.

Color Profile Basic Concepts

The first step in the color profile process is loading raster data from an external location (file, memory, or other) and converting it from some graphical file format to an internal uniform format that is incorporated into the `HIGEAR` object. After the image is loaded into memory, it is possible to perform different manipulations with it such as image processing transforms, color conversions, displaying, printing, and, finally, an export operation that converts raster data from internal representation into an external graphics file format.

All these steps may requre color conversions. As the color profile is always associated with the appropriate device, there are several *virtual devices* defined in ImageGear. These "virtual devices" are not real devices, but just abstract things that have associated color spaces and are used to convert color data from one color space to another. There are three virtual devices defined in ImageGear:

- **import devices** - used for all import operations. Such import operations as loading a raster image using the format filter assume that imported color data is dependent on the import device. In other words, any format filter that loads color data in a device-dependent format from an external file format assumes that this data is dependent on this specific device.

- **export devices** - the same as the import device, but used for export operations.
- **working devices** - associated with color data stored in a `HIGEAR` object.

So we can assume every color conversion is a transition from one device to another. For example, loading an image from an external file using the file format filter (import device) to the internal `HIGEAR` representation (working device) assumes that the color data has to be “copied” from the import to the working device, i.e. color data has to be converted using color profiles associated with import and working devices.

ImageGear supports different color spaces such as RGB, CMYK and grayscale. So each virtual device may be used with different color spaces. For example, the format filter may load pixels from an external file format in CMYK color space, and it will be necessary to convert color data from the CMYK color space associated with the import device to CMYK or another color space associated with the working device. Each device may have associated color profiles for different color spaces, and they are organized into groups - one group for each device.

You can find the exact definition of the term “color profile” in public specification ICC.1:1998-09 or newer. In general a color profile consists of a set of objects and transforms the specified parameters of color conversion from a standard device-independent color space (PCS) to a necessary device-dependent one. The ImageGear color management system accepts a color profile in the format specified in the ICC.1:1998-09 specification and converts it into internal representation for faster processing.

ImageGear Color Profile Groups

ImageGear allocates 3 color profile groups:

- ICP (Import Color Profile)
- ECP (Export Color Profile)
- WCP (Working Color Profile)

ImageGear allows you to set and get the actual value of a color profile in every group. In spite of the fact that operations with color profiles from different groups are very similar, at a low level there are some differences in how profiles from different groups are processed.

ICP (Import Color Profile)

This group of profiles is used during a filter load operation. In some cases a raster image file contains device-dependent color data such as RGB or CMYK but does not have a color profile associated with it. So this color data can be interpreted in a different manner depending on the color profile used. The ICP profile of appropriate color space should be used in this case. However, if a raster image contains a color profile, this one is used instead of the standard ICP profile.

ECP (Export Color Profile)

This group of profiles is very similar to ICP but is used in the filter export operation. If the raster image file to be exported does not allow you to store the color profile or if it needs to be stored in a specific color space, ECP allows you to provide the profile, and the output image will be converted to that color space.

WCP (Working Color Profile)

This group of color profiles provides information about the default color global parameters used to represent the color data for `HIGEAR` objects. Those global parameters are used if the image does not have a local color profile associated with it.

When color processing is performed on an image, either a local or global color profile is used. When an image is exported or imported, its pixel data, converted from one color space to another, is described by the WCP associated with the image and corresponding profile from the ECP or ICP group.

Color Profile Manager API

There are public functions implemented in ImageGear that allow you to set and get actual values for each group of color profiles.

To start working with the color profile manager, it is necessary to activate it. By default, the color profile manager is disabled in ImageGear, and all color-related operations are exactly the same as in previous versions (prior to ImageGear). There is a Boolean global parameter named `CPM.ENABLE_PROFILES` used to control color profile management.

To activate color profile management, call the ImageGear global control function (see also the section Working with Global Control Parameters:

```
AT_BOOL     bEnable = TRUE;
IG_gctrl_item_set( “CPM.ENABLE_PROFILES“, AM_TID_AT_BOOL,
&bEnable, sizeof(AT_BOOL), NULL );
```

Another thing related to color management is that additional files are necessary - color profile files to be used as default profiles. By default those files should be located in the same folder as ImageGear and its components are located, and those files should be named as “`ig_rgb_profile.icm`“ and “`ig_cmyk_profile.icm`“ respectively for RGB and CMYK color spaces. Correspondent global parameters `CPM.RGB_PROFILE_PATH` and `CPM.CMYK_PROFILE_PATH` exist to specify the full path to default profiles. Their values can be changed using the function **`IG_gctrl_item_set()`**, like below in the example:

```
CHAR     profile[256];
strcpy( profile, “d:\\profile\\rgb_profile.icm“ );
IG_gctrl_item_set( "CPM.RGB_PROFILE_PATH",
AM_TID_MAKELP(AM_TID_CHAR), profile, strlen(profile)+1, NULL );
```

After color profile management is activated, it is possible to use the next functions to set color profiles in global parameters or to a specific image and to get information about currently used profiles.

The function:

```
IG_cpm_profile_set(AT_MODE nColorSpace, DWORD nProfileGroup,
LPBYTE lpRawData, DWORD dwRawSize, AT_BOOL  bConvert);
```

allows you to set a new color profile value (`lpRawData`) to a group specified by the second parameter and associate it with the color space given by the first parameter. `lpRawData` is a pointer to the memory buffer that contains the color profiles in valid ICC format, and if it is `NULL` then the default color profile will be set. The last parameter specifies how to process the color data of the images associated with it. If it is `TRUE` then all images associated with the color profile will be converted to a new color format.

The function:

```
IG_cpm_image_profile_set(HIGEAR hIGear, LPBYTE lpRawData,
DWORD dwRawSize, AT_BOOL  bConvert)
```

is a special case of a working profile set operation that associates a given color profile with a single image. In this case the image becomes associated not with a global working profile but with a locally given profile.

The function:

```
IG_cpm_profile_get(AT_MODE nColorSpace, DWORD nProfileGroup,
LPCHAR pStatusStr, UINT nStatusSize, LPUINT lpnStatusLen,
LPDWORD lpnProfileSize, LPBYTE lpProfileData, DWORD
dwProfileDataSize)
```

can be used to get the current value of a color profile associated with the color space given by the first parameter in the profile group specified by the second parameter. Other parameters return information about the profile along with the profile itself.

To get the color profile information associated with a given image, the function

```
IG_cpm_image_profile_get(HIGEAR hIGear, LPAT_BOOL lpbIsLocal,
LPAT_MODE lpnColorSpace, LPCHAR  lpStatusStr,UINT  nStatusSize,
LPUINT lpnStatusLen, LPDWORD lpnProfileSize, LPBYTE
lpProfileData,
DWORD  dwProfileDataSize)
```

can be used. It returns information about the local profile if it is associated with an image, or a global profile otherwise. The second parameter returns `TRUE` if the local profile is associated or `FALSE` if it is not.

The function:

```
IG_cpm_profiles_reset(AT_BOOL bConvert)
```

is introduced to reset all global profiles to their default values. If `bConvert` is `TRUE`, then all images associated with the old previous profiles will be converted to the new ones.

For a detailed description of the color profile functions, please see the "Core Component API Function Reference" chapter.

Working with Global Control Parameters

ImageGear provides you with a set of API functions that allow you to add new global control parameters to your ImageGear application, set new values for already existing global control parameters, and retrieve information about these parameters.

To add a new global parameter as well as to set new values for existing parameters use the function:

```
IG_gctrl_item_set(LPCHAR ControlID, AT_MODE nValueType, LPVOID
lpValue, DWORD dwValueSize, LPCHAR lpTextInfo)
```

This function will search for the global parameter specified by the `ControlID` name (syntax: "`<GRPNAME>.<Param name>`"), and if it is found will set a new value for it. If it is not found the function will add this new parameter to the global control parameters list. Through the `lpTextInfo` argument you can also set the text description of the specified global parameter.

To retrieve information about the global control parameter use the functions:

```
IG_gctrl_item_get(LPCHAR CtrlID, LPAT_MODE lpnValType, LPVOID
lpValue, DWORD dwValSize, LPDWORD lpdwValSize, LPCHAR
lpTextInfo, DWORD dwTextBufSize, LPDWORD lpdwTextInfoSize)
IG_gctrl_item_by_index_get(UINT nIndex, LPCHAR CtrlID, DWORD
dwIDSize, LPAT_MODE lpnValType, LPVOID lpValue, DWORD dwValSize,
LPDWORD lpdwValSize, LPCHAR lpTextInfo, DWORD dwTextBufSize,
LPDWORD lpdwTextInfoSize );
```

The first function returns the value and the text description of the global control parameter specified by name. The second function returns information about the control parameter specified by its index in the global parameters list. Both functions return `FALSE` if the specified global parameter is not found.

If you want to know the general amount of global control parameters currently existing in the global parameters list, call the function:

```
IG_gctrl_item_count_get();
```

If you need to know an index of the global control parameter in the parameters array, use this function:

```
IG_gctrl_item_id_get (UINT nIndex, LPCHAR lpCtrlID, UINT
nBufSize);
```

Working with Color Space Conversion Engine

Color space conversion available for separate raster as well as for entire image. The simplest version may be used to convert entire image. For these purposes **`IG_image_colorspace_convert()`** routine must be used. It takes image, new color space identifier and conversion options (optionally) as incoming parameters. It is necessary to note that channels structure of converted image may differ from initial one. The most common changes are quantity of channels and depth for channels. Quantity of channels may be changed because of differences in color space and because of changes in extra channels (controlled by conversion options). Changes in depth usually occur if source channels have different depths. E.g. If source image has 5-6-5 bits per pixel, converted image will have 6-6-6 channel depths.

To provide advanced control for color space conversion, `AT_COLORSPACE_CONVERSION_OPTIONS` structure has been added. Below is the explanation of each attribute:

- `IsRaw (BOOL)` specifies should image content be converted or just changed value of color space id. That is if `IsRaw` is false, conversion flows in regular way with modification of pixels using color space conversion rules. If

parameter is TRUE, in this case only value of attribute that stores color space id is changed without applying conversion to image data. It's necessary to note that raw conversion is not available for all cases. E.g. conversion from IG_COLOR_SPACE_ID_Gy to IG_COLOR_SPACE_ID_RGB with IsRaw = TRUE is not available because of absence of sufficient channels quantity.

- AlphaBlending is designed to control conversion in case of alpha channel presence. If neither alpha nor premultiplied alpha channel is available in incoming image, this attribute is not used. But if source image has any alpha and new color space has no alpha bit (neither IG_COLOR_SPACE_ID_A nor IG_COLOR_SPACE_ID_P), AlphaBlending.Apply specifies whether or not the image should be blended with AlphaBlending.Background color. That is, if Apply attribute is equal to FALSE, alpha channel will simply be cut. Otherwise image will be blended with background according to alpha channel values.
- ExtraChannels provides additional control of how extra channels should be transferred. If its Range attribute is equal to FALSE, then all extra channels are transferred from source to converted image (if new color space have IG_COLOR_SPACE_ID_Ex bit. otherwise all extra channels will be cut). Otherwise StartChannel and ChannelCount attributes specify the range of extra channels to be transferred from source to result.
- ColorConverter represents a custom converter for pixel color part. For now, this is a reserved attribute and must be NULL.

Advanced color space conversion routines allow operating on not only images but particular rasters. It's also possible to supply a custom channels structure to produce conversion data with necessary channel depths.

To start operating conversion, HIGCSCONVERTER instance must be created. Any of **IG_colorspace_conversion_create()**, **IG_colorspace_conversion_create_from_image()** or **IG_colorspace_conversion_create_from_dib_info()** may be used for this purpose. First variation has eight incoming parameters and one output:

- sourceSpace (enumIGColorSpaceIDs) - identifier of color space from which conversion will occur.
- sourceSpaceAttribs (const AT_RGBQUAD*) - represents palette entries. Must be specified if source color space is indexed. Otherwise may be NULL.
- sourceChannelDepths(const AT_INT*) - array of channel depths in source image or raster.
- sourceChannelCount (AT_INT) - quantity of channels in source image or raster.
- resultSpace (enumIGColorSpaceIDs) - identifier of color space to which image or raster will be converted.
- resultChannelDepths (const AT_INT*) - channel depths for result image (raster). May be NULL - in this case, resultChannelDepths will have same values as output of **IG_colorspace_conversion_result_channels_build()** routine.
- resultChannelCount (AT_INT) - quantity of channels in result image or raster. Ignored if resultChannelDepths is NULL.

- `options` (`LPCAT_COLORSPACE_CONVERSION_OPTIONS`) - conversion control parameters.

`IG_colorspace_conversion_create_from_image()` and **`IG_colorspace_conversion_create_from_dib_info()`** are designed for convenience and retrieve source attributes (space, palette, channels) from supplied image or DIB information.

Once color space converter was allocated it may be used in any of **`IG_colorspace_conversion_apply_to_image()`** and **`IG_colorspace_conversion_apply_to_raster()`** routines.

To release conversion, **`IG_colorspace_conversion_destroy()`** routine must be used.

It's necessary to note that advanced conversion routines may be used not only for color space conversion, but for channel depth change too (even if `IsRaw` attribute of conversion options turned on). It's possible to provide desired channels structure and the converter will produce output according to specified structure.

Auxiliary routines of advanced colorspace conversion are **`IG_colorspace_conversion_result_palette_get()`** and **`IG_colorspace_conversion_result_channels_build()`**.

`IG_colorspace_conversion_result_palette_get()` is designed to produce palette if new color space is indexed. But note that conversion to `IG_COLOR_SPACE_ID_I` color space is possible only from `IG_COLOR_SPACE_ID_I` and `IG_COLOR_SPACE_ID_Gy` with depth not greater than eight.

`IG_colorspace_conversion_result_channels_build()` may be used to get default channels structure for converted data. The result of this routine is used by conversion routines if `resultChannelDepths` parameter has `NULL` value.

Working with Grayscale Look Up Tables

ImageGear provides a set of functions for working with grayscale LUTs. A grayscale LUT object is represented as an opaque handle: `HIGLUT`. Use the `IG_LUT_…` group of functions to create, destroy and access features of grayscale LUTs.

An `HIGLUT` object can have various input and output depths. Both input and output can be signed or unsigned.

The ImageGear Medical component provides a set of API functions that allows you to build grayscale LUTs according to various DICOM display settings.

Example:

```
HIGLUT GrayLUT;
AT_INT index;

// Create a LUT
IG_LUT_create(12, TRUE, 8, FALSE, &GrayLUT);

// Fill the LUT with a linear table, transforming 12-bit signed
image to 8-bit unsigned
for (index = -2048; index<2048; index++)
{
```

```
        value = (index + 2048) / 16;
        IG_LUT_item_set(GrayLUT, index, value);
    }

    IG_image_grayscale_LUT_update_from(hIGear, GrayLUT);
```

ImageGear allows attaching grayscale LUTs to images and to image display settings.

See Also:

Grayscale Look Up Tables

Working with Multi-page Images

Along with a single-page image handle (`HIGEAR`), ImageGear provides support for multi-page images. The new handle, `HMIGEAR`, represents an array of single-page images. You can now use ImageGear to:

- Create and delete an internal representation of a multi-page image (`HMIGEAR` handle),
- Open and associate a multi-page image file with an external file,
- Access and manipulate pages within the multi-page image,
- Manipulate pages in the external image file, such as loading, saving, swapping and deleting pages,
- Retrieve information about multi-page images and about associated external files.

The list of functions are divided into two categories:

- Those that work with external associated multi-page files. These functions appear as `IG_mpf_...`.
- Those that deal with multi-page image arrays. These functions begin with `IG_mpi_`.

For example, the function **`IG_mpf_page_swap()`** swaps pages in the external file.

Creating and Deleting a Multi-page Image

An internal representation of a multi-page image is an object of the data type `HMIGEAR`. This data type encapsulates an array of pages, which are objects of type `HIGEAR`. All `IG_mpi_***` functions are introduced to access and manage this array.

The first step when working with a multi-page image is to allocate and initialize an object of type `HMIGEAR`. To do this, call the function **`IG_mpi_create()`**. This function creates a new multi-page image and sets the page array size to the given number of pages. Pages are numbered from 0 to `nPageCount`-1, where `nPageCount` is the current size of the page array and the default page value is `NULL`.

A multi-page image is an array of pages. It is not necessary that all elements of this array contain valid HIGEAR objects. Some pages may have a NULL value and the function **IG_mpi_page_is_valid()** can be used to quickly identify whether a page with a given number contains a valid HIGEAR image. The function **IG_mpi_page_get()** retrieves the value, and the **IG_mpi_page_set()** function replaces a page value with a given index in the page array of a multi-page image.

The function **IG_mpi_page_count_get()** returns the current size of a page array and the function **IG_mpi_page_count_set()** changes then size of the page array.

When a multi-page image is no longer needed, you should delete it using the **IG_mpi_page_delete()** function. This function deletes all valid pages in the array and frees all memory allocated in the HMIGEAR handle.

See Also:

Getting Information and Sorting Images

Opening and Closing an External Image File

After a multi-page image is created it can be associated with external image. This allows you to set a relationship between a page array of a given multi-page image and pages of a multi-page file. After making that association, but before closing the file, ImageGear stores format-related information in memory. This allows you to perform page manipulation operations, such as page loads, saves, swaps, or deletes - quickly, without scanning all of the format structure of the associated file image. This may be useful if the number of pages in the file is large. It also improves the performance of these operations.

Use the function **IG_mpi_file_open()** to open or create an external multi-page file and associate it with the multi-page image given by the HMIGEAR object. Use a third parameter of this function to specify two open modes. If IG_MP_OPENMODE_READONLY is specified, then the file opened has read-only access. only these operations are allowed. They need not change the file (for example, page loading).

If IG_MP_OPENMODE_READWRITE is used, then the file is opened with read-write access and all supported operations are allowed. Not all format filters support such operations, such as page insert, delete, and swap. The function **IG_fltr_info_get()** can be used to get information about all implemented features of some particular format.

The third parameter of this function provides the format identifier (one from defined IG_FORMAT_… constants in accucnst.h) and is used only when the file image opens in IG_MP_OPENMODE_READWRITE mode. The file does not exist and it is ignored in other cases. This parameter is used to specify the format of the image to be created.

Example 1:

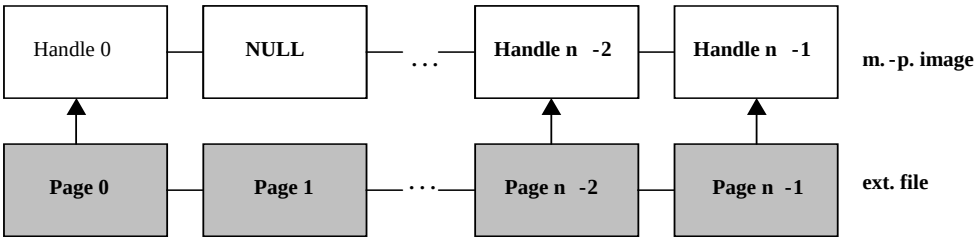
```
#include “accucnst.h”
 …
HMIGEAR hMIGear;        /* HMIGEAR handle returned   */
AT_ERRCOUNT nErrCount;     /* number of errors reported */
 …
nErrCount = IG_mpi_create(&hMIGear, 0);

if (!nErrCount)
```

```
{
nErrCount = IG_mpi_file_open(“picture.tif”, hMIGear,
IG_FORMAT_UNKNOWN,
IG_MP_OPENMODE_READONLY);
        …
   nErrCount = IG_mpi_close( hMIGear );
}else
       /* error handling */
 …
```

If this function is opened for read-only access, then page array of multi-page images is set to the number of pages that is equal to number of pages in the external file. If the file is opened for read-write access, then the multi-page image is not changed.

Use the function **`IG_mpi_close()`** to disassociate a multi-page image from an external file and close the file and free all correspondent resources.



Loading and Saving Pages

After the multi-page image is associated with an external file, it is possible to perform operations such as page loads, saves, swaps, and deletes. If open mode is read-only, then only the page load is allowed. All others will be set to an error.

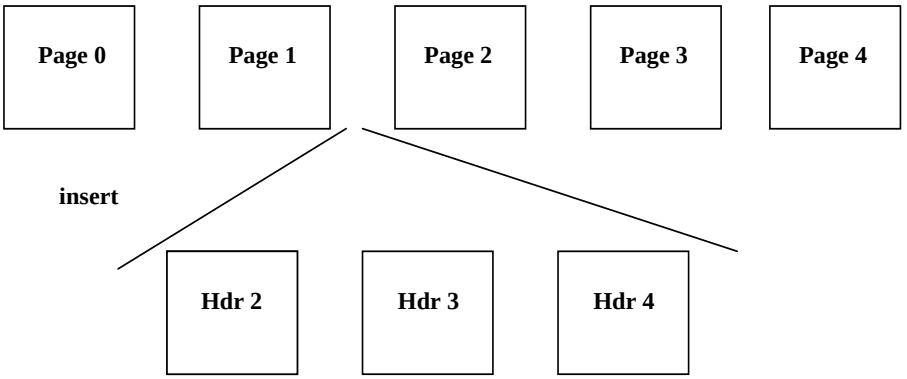
Use the function **`IG_mpf_page_load()`** to load pages from an external file into a multi-page image. The second argument of the function is a zero-based index of the first page for loading. The third argument specifies the number of pages to load starting from this index. The order and location of the loaded pages in the page array is the same as in the external file. Therefore, if the file pages start from `nFirstIndex`, then it loads into the page array starting with `nFirstIndex`. If necessary, the page arrays are expanded to fit all of the requested number of pages. If, while loading, some elements of the page array contains a valid `HIGEAR` image, then it is not deleted. It is then assigned a new value of the image loaded from the file.

Use the function **`IG_mpf_page_save()`** to save pages from a multi-page image into an external file. The second and third arguments are the same and have the same meaning as the second and third arguments for the load function. The fourth argument specifies the compression method (for example, `IG_COMPRESSION_JPEG` or `IG_COMPRESSION_LZW` for TIFF image) and applies to all pages from the given range. The last parameter of this function specifies how to save the pages into a file. There are two modes are possible

- IG_MPF_SAVE_INSERT
- IG_MPF_SAVE_REPLACE

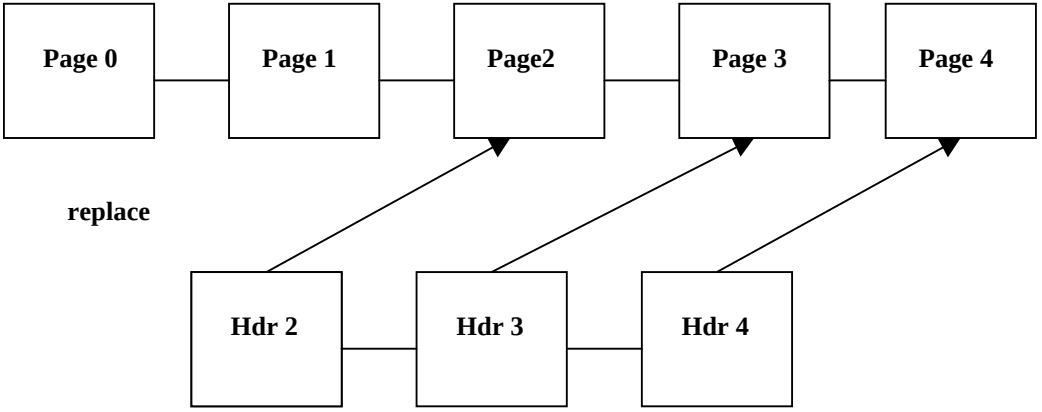
The first mode insert pages into the file at the location specified by second parameter, which is demonstrated in the following picture:

IG_MPF_SAVE_INSERT:



The second mode replaces pages starting from the index specified by the second parameter:

IG_MPF_SAVE_REPLACE case:



Example 2:

To load pages 2 through 5 from the file “image1.tif” and insert them into file “image2.tif,” the program is:

```
/*…*/
HMIGEAR hMIGear;
AT_ERRCOUNTnErrCount;
UINT   nStartPage;
UINT   nCount;

nErrCount = IG_mpi_create(&hMIGear, 0);
if (!nErrCount)
{
nErrCount = IG_mpi_file_open( “picture1.tif”, hMIGear,
IG_FORMAT_UNKNOWN,
IG_MP_OPEN_READ );
       nStartPage = 2;
       nCount = 4;
       nErrCount = IG_mpf_page_load( hMIGear, nStartPage, nCount
);
}
if (!nErrCount)
{
       nErrCount = IG_mpi_close( hMIGear );

       if (!nErrCount)
          nErrCount = IG_mpi_file_open( “picture2.tif”,
hMIGear, IG_FORMAT_UNKNOWN, IG_MP_OPEN_READWRITE );
   if (!nErrCount)
   nErrCount = IG_mpf_page_save( hMIGear, nStartPage, nCount,
IG_COMPRESSION_JPEG, IG_MPF_SAVE_INSERT );
}
if (nErrCount)
{
       /* error handling */
#include “accucnst.h”
}
```

NOTE: If a page from the pages to save is not a valid `HIGEAR` image, then this page is ignored during the save operation.

NOTE: IG_MPF_SAVE_REPLACE mode is acceptable for filters that support the page deletion operation. This information can be obtained using the function **IG_fltr_info_get()** and it should return the IG_FLTR_PAGEDELETESUPPORT flag, which set in the dwInfoFlags parameters.

Using Other Functions that Work with Pages

When an external file is opened in read-write mode, an operation such as page delete and page swap is possible. Not all filters support it. It is necessary to check with the function **IG_fltr_info_get()** for flags IG_FLTR_PAGEDELETESUPPORT and IG_FLTR_PAGESWAPSUPPORT in the lpdwInfoFlags parameter.

IG_mpf_page_swap() is used to reorder pages in the external file. The second and third arguments of this function are zero-based indexes of the pages to be swapped.

The **IG_mpf_page_delete()** function deletes the specified pages from the external file. The index of the first deleted page is passed through the second argument and number of pages to be deleted is passed through the third argument.

The example below shows how to reorder pages in the multi-page file.

Example 3:

```
AT_ERRCODE mpfPageReorder(
    HMIGEAR hMIGear
)
{
    UINT nPageCount;
    AT_MODE nFormatID;
    AT_ERRCOUNT nErrCnt;
    DWORD dwInfoFlags;
    UINT i;

IG_mpf_info_get( hMIGear, &nFormatID );
    if( nFormatID==IG_FORMAT_UNKNOWN )
    return-1; /* file is not associated */

    IG_fltr_info_get( nFormatID, &dwInfoFlags, NULL, 0, NULL, 0,
NULL, 0 );
    if( (dwInfoFlags&IG_FLTR_PAGESWAPSUPPORT)==0 )
    return-1;/* format filter does not support the page swap
operation */

    nErrCnt = IG_mpf_page_count_get( hMIGear, &nPageCount );

    for( i = 0; (i < nPageCount/2) && (nErrCnt==0); i++ )
nErrCnt = IG_mpf_page_swap( hMIGear, i, nPageCount – i - 1 );

    return -nErrCnt;
```

```
}
```

The example below demonstrates how to perform this operation on a multi-page image located in memory:

Example 4:

```
AT_ERRCODE mpiPageReorder(
HMIGEAR hMIGear

{
HIGEAR hPage1, hPage2;
UINT i, nPageCount;
AT_ERRCOUNT  nErrCnt = 0;

nErrCnt = IG_mpi_page_count_get( hMIGear, &nPageCount );

for( i = 0; (i<nPageCount/2) && (nErrCnt==0); i++ )
   {
   nErrCnt = IG_mpi_page_get( hMIGear, i, &hPage1 );
   nErrCnt += IG_mpi_page_get( hMIGear, nPageCount – i - 1,
&hPage2 );
   nErrCnt += IG_mpi_page_set( hMIGear, i, hPage2 );
   nErrCnt += IG_mpi_page_set( hMIGear, nPageCount – i - 1,
hPage1 );
}
return -nErrCnt;}
```

See Also:

`IG_fltr_pageswap_file()`

`IG_fltr_pagedelete_file()`

Getting Information and Sorting Images

Using the Multi-page Image Callback Function

Multi-page image operations, implemented by the above functions, support a notification mechanism that allows you to track information about when and how the multi-page image or associated file was changed. This can be done using the functions **`IG_mpi_CB_set()`**, **`IG_mpi_CB_get()`**, **`IG_mpi_CB_reset()`**, and **`IG_mpi_CB_reset_all()`**.

Use the function `IG_mpi_CB_set(hMIGear, lpPrivate, lpfnUpdate, lpdwCBID)` to associate new callback data with the given `hMIGear` handle. The second parameter, `lpPrivate`, is any `LPVOID` pointer that the callback function `lpfnUpdate` receives through the second parameter. `lpfnUpdate` is a pointer to the function that implements the following interface:

```
typedef  VOID (LPACCUAPI LPFNIG_MPCB_UPDATE)(
   DWORD dwCBID,
   LPVOID lpPrivate,
   AT_MODE nMode,
   UINT nPage,
   UINT nCount
   );
```

The last parameter, `lpdwCBID`, is a pointer to that application which receives a unique DWORD identifier for the associated callback data. This ID is used to delete callback data using the function `IG_mpi_CB_reset()` and retrieves callback data using the function `IG_mpi_CB_get()`. After the `IG_mpi_CB_set()` function is executed, ImageGear calls the `lpfnUpdate` function every time a multi-page image is changed. This allows the application to react accordingly, and updates the related objects such as GUI windows. As soon as this callback function is called from the context of the thread that performed the operation, its execution is blocked until the callback function is complete. From one side, this can be used for synchronization; but from another side, it should be used carefully so that it does not affect performance.

By calling the callback function, ImageGear passes the type of changes through the `nMode` parameter. The sense of `nPage` and `nCount` parameters depend upon `nMode`. The following table list all possible cases:

nMode	nPage	nCount	Description
`IG_MPCBMODE_MPI_DELETE`	Not used	Not used	Notifies the application that a multi-page image is going to be deleted.
`IG_MPCBMODE_MPI_ ASSOCIATED`	Not used	Not used	Notifies the application that a multi-page image is just associated with external file.
`IG_MPCBMODE_MPI_CLOSE`	No used	Not used	Notifies the application that a multi-page image is going to close associated external file
`IG_MPCBMODE_MPI_CB_SET`	Not used	Not used	Notifies the application that this callback data is just set. This notification receives only the callback function that just has been set.
`IG_MPCBMODE_MPI_CB_ RESET`	Not used	Not used	Notifies the application that this callback data is to be reset.
`IG_MPCBMODE_MPI_ PAGEINSERTED`	index of where new pages starts	number of new pages inserted	Notifies the application that new pages are inserted into the multi-page image.
`IG_MPCBMODE_MPI_ PAGEUPDATED`	index of the first updated page	number of updated pages starting from nPage	The application updated pages in the multi-page image
`IG_MPCBMODE_MPI_ PAGEDELETED`	first deleted page index was have	number of deleted pages	The application deleted pages in the multi-page image.

IG_MPCBMODE_MPF_PAGEINSERTED	index of where new pages starts	number of new pages inserted	The application inserted new pages into external file image.
IG_MPCBMODE_MPF_PAGEUPDATED	index of the first updated page	number of updated pages starting from nPage	The application updated pages in the associated external multi-page image file.
IG_MPCBMODE_MPF_PAGEDELETED	Index of the first deleted page	number of deleted pages	The application deleted pages in the associated external multi-page image file

When the application does not need to receive any more information from the callback data, it should call the function **IG_mpi_CB_reset**(hMIGear, dwCBID), where dwCBID is a unique identifier of the association returned by **IG_mpi_CB_set()**.

The function **IG_mpi_CB_reset_all()** removes all callback data associated with all previously allocated identifiers.

See Also:

Processing of non-image data through filter callback functions

Working with ImageGear Callback Functions

Working with Image Utility Functions

ImageGear's image utility family of functions provides you with the capabilities to create, import, and export images in either DIB or DDB format, and to obtain information about any image for which you have a HIGEAR handle. You can also obtain information about image files stored on mass storage devices, such as the file format type, compression, width, height, bits per pixel, or number of pages if a multi-page file.

A few special purpose image utility functions tell you whether a HIGEA R variable contains a valid image handle, and whether an image is gray-scale. And there are six functions to help you read and write palettes, either whole or one entry at a time, and save them to disk. There's also a function that lets you set special control options that alter the operation of ImageGear's file format read-write filters during file operations.

One additional important image utility function sets an image's "image rectangle", which determines the portion of the image to be displayed, printed, or saved during display, print, and save operations. Refer to "Core Component API Function Reference" Chapter for detailed calling sequences and further notes on these functions.

Creating DIBs and DDBs

- To create a new HIGEAR with an empty DIB, use the following code:

```
IG_image_create_DIB_ex ( nWidth, nHeight, nBpp, lCompression,
lpDIB = NULL, &hIGear );
```

Later, you could use the IG_DIB_...() for direct pixel access functions, or an IG_IP_blend_ …() function to create an image bitmap. You can use the IG_palette_ …() functions (described later in this section) to add a palette to the DIB.

- To create a new HIGEAR with a DIB filled by copying an existing DIB, use the same function as demonstrated above, but call it like this:

```
IG_image_create_DIB_ex ( 0, 0, 0, 0, lpDIB, &hIGear );
```

In this example, lpDIB is a pointer to the existing DIB to copy, and the first four arguments are ignored. Width, height, bits per pixel, and “compression” will be copied from the existing DIB, along with its image bitmap and palette.

- To give a HIGEAR handle to an existing DIB, use the import function:

```
IG_image_DIB_import ( lpDIB, &hIGear );
```

This call returns you to the HIGEAR handle assigned to your DIB.

- To create a HIGEAR whose image is a copy of the image in an existing DDB:

```
IG_dspl_DDB_import ( hBitmap, hPalette, &hIGear);
```

In this call, you provide the DDB’s HBITMAP handle, and optionally the handle of a palette. ImageGear creates a DIB for you and returns the new DIB’s HIGEAR handle.

- To create a DDB whose image is a copy of the image in an existing HIGEAR:

```
IG_dspl_DDB_create( hIGear, IG_GRP_DEFAULT, hDC, nWidth,
nHeight, TRUE, &hBitmap, &hPalette )
```

Provide the HIGEAR handle and the width and height for your DDB as well as the addresses of an HBITMAP and an HPALETTE variable to receive the DDB and palette handles. Because this function belongs to the display group, there should be a group identifier, nGripID, to specify where to get the options needed to complete this operation. hDC is necessary to specify the color format output to which the DDB should be compatible.

Deleting DIBs and DDBs

- To delete the HIGEAR, but keep the DIB in existence (and obtain its address):

```
IG_image_DIB_export( hIGear, lpDIB);
```

The DIB’s address is returned in your LPAT_DIB variable lpDIB.

- To delete the HIGEAR and the DIB, but produce a copy of the image in DDB format:

```
IG_dspl_DDB_create( hIGear, IG_GRP_DEFAULT, hDC, nWidth,
nHeight, TRUE, &hBitmap, &hPalette )
```

Provide the HIGEAR handle of the image to delete, and provide the addresses of the HBITMAP and HPALETTE variables to receive the handles for the created DDB and palette.

- To delete a HIGEAR, including the DIB, when you are entirely done using use the following call:

```
IG_image_delete ( hIGear );
```

Note that in all the above cases, ImageGear will not release memory that it did not allocate. For example, assume your application has allocated memory, created a DIB, and subsequently used an **IG_image_DIB_import()** call to give this DIB a HIGEAR handle. In this case, a later call to IG_image_delete() will delete the HIGEAR structure but will not free the DIB's memory. The owner of the DIB's memory (in this case, your application) would issue a Windows DeleteObject() call to free this memory.

Reading and Writing Palettes

There are six image utility functions that read and write palettes in memory, or load and save them between memory and disk. To transfer a palette between your own memory area and a HIGEAR image's DIB (that is, to get or set the DIB palette), use:

```
AT_RGBQUAD palette[256];
IG_palette_get ( hIGear, palette );
IG_palette_set ( hIGear, palette );
```

In the above calls, lpPalette should point to the first of an array of AT_RGBQUAD structures, one structure per palette entry.

If instead you want to move just one entry to or from the DIB palette:

```
AT_RGB    rgbPaletteColor;
IG_palette_entry_get ( hIGear, &rgbPaletteColor, nIndex );
IG_palette_entry_set ( hIGear, &rgbPaletteColor, nIndex );
```

In the single entry calls, you supply a pointer to a single structure of type AT_RGB. The third argument has a value between 0 and 255, specifying which palette entry to get or set (Remember that an AT_RGBQUAD structure consists of 4 bytes ordered Blue-Green-Red-Unused(0), while an AT_RGB struct consists of 3 bytes ordered Blue-Green-Red.).

To load and save palettes between memory and disk, use the functions:

```
IG_palette_load (“filename”, palette, nEntries, bOrder,
lpFileType);
IG_palette_save (“filename”, palette, nEntries, lpFileType);
```

See the descriptions of the above functions in "Core Component API Function Reference", for details on specifying the arguments.

Getting Information about a HIGEAR Image

To find out if a HIGEAR variable currently holds a valid handle:

```
if ( IG_image_is_valid(hIGear) )  { … }
```

Similarly, to find out if it is a gray-scale image:

```
if ( IG_image_is_gray(hIGear) )  { … }
```

To obtain the width, height, and bits per pixel of an image, or the DIB compression type, use (respectively):

```
AT_DIMENSION nWidth, nHeight;
UINT nBpp;
```

```
IG_image_dimensions_get ( hIGear, &nWidth, &nHeight, &nBpp );
DWORD nCompression;
IG_image_compression_type_get ( hIGear, &nCompression );
```

NOTE: Be careful to declare the types as `AT_DIMENSION` and `DWORD` where shown above (rather than `INT` or `UINT`). On some development platforms, `AT_DIMENSION` and `DWORD` are not the same size as `INT`.

And to learn the position and size of an image's image rectangle, use **`IG_dspl_layout_get()`** API.

NOTE: Here also be careful to use the correct type**:** `AT_RECT`, not Windows structure type `RECT`, whose fields may be a different length.

To set the image rectangle, use the function **`IG_dspl_layout_set()`**. Examples are provided in the sections Displaying Images and Saving Images .

If you need to access a DIB directly, refer to the descriptions of function **`IG_image_DIB_palette_pntr_get()`**in "Core Component API Function Reference".

Obtaining Information about Files

Three sets of image utility functions obtain information about files. The `IG_info_get …()` functions (namely: **`IG_info_get_ex()`**, **`IG_info_get_FD_ex()`**, and **`IG_info_get_mem_ex()`**) return to you the file format type and file compression type, and the DIB header that would be created if you loaded the file (giving you width, height, and bits per pixel, for example).

The similarly named `IG_page_count_get …()` and `IG_tile_count_get …()` functions tell you how many pages a file has if it is a multi-page file, and how a given page is tiled if the file format supports tiled pages. See the descriptions of these functions in Core Component API Function Reference. For more discussion about tiles, see also the section entitled Working with Tiled Images.

For all three sets of functions use:

- `IG_ …get()` form (such as `IG_page_count_get()`) if you are referencing the file by its filename.
- `IG_ …get_FD()` form if the file is already open and you are referencing it by its File Descriptor handle.
- `IG_ …get_mem()` form if the file image is in memory and you are referencing it by its memory address.

See Also:

Getting Information and Sorting Images

Working with Library Utility Functions

ImageGear's small group of functions, called "library utility" functions, are provided so that you can conveniently obtain the ImageGear version and function status. The functions in this group have names beginning with `IG_version_ …()`, `IG_error_ …()`, `IG_err_ …()`.

For instructions on how to use these functions, see the section entitled Detecting and Handling Errors . Detailed information on these functions is provided in Core Component API Function Reference.

Detecting and Handling Errors

Handling error conditions detected inside ImageGear functions is easy, because ImageGear functions handle all errors in a single, uniform way. Even low-level ImageGear functions that you can not call directly, handle errors in this same way.

When an error condition is detected by an ImageGear function, the function places an error code indicating specifically what happened, along with information telling where the error occurred, in an internal memory area called the ImageGear error stack. This error stack remains available to your application as it executes, so you can inspect and treat the errors where (in your program code) you wish, and as you wish. After placing the error or errors on the error stack, the ImageGear function returns to its caller (returning the count of errors now on the stack, if the function's return-type is `AT_ERRCOUNT`).

More than one error may be placed on the error stack as a result of a single ImageGear function call. This is because an ImageGear function that you call will often call lower level ImageGear functions (not directly callable by your application). Each such lower level function may itself place an error onto the error stack before returning to its caller. Upon return to your application, there may be several errors on the stack. Note that in such a case, the lowest level function's error was placed on the stack first, and the highest level function (the one that your application called directly) placed its error on the stack last, just before returning to your application because it could not proceed (due to the error).

ImageGear provides four basic functions for accessing the error stack. They are:

`IG_error_clear()`	Clear the error stack
`IG_error_check()`	Obtain the number of errors on the error stack
`IG_error_get()`	Get a specified error from the error stack
`IG_error_set()`	Place an error on the error stack. This function can be useful for the forced termination of loading or saving procedures if you have defined your own callback routines.

The example below shows how you may use some of the above functions to display in a Message Window on your screen every error that is on the ImageGear error stack. Refer to descriptions of these functions in Chapter "Core Component API Function Reference" for thorough explanations of their arguments.

```
HIGEAR hImage; /* Will hold handle of image loaded below */
```

```
INT i; /* Will hold Loop Index and Error Index */
INT iLineNumber; /* Will hold returned Line Number */
BYTEszFileName[30]; /* Will hold ret'd module name, <= 29 chars
*/
INT cbFileNameSize; /* Will hold size of szFileName array */
AT_ERRCODE  iCode; /* Will hold returned ImageGear Error Code */
AT_ERRCOUNT  nErrcount; /* Will hold count of errors on error
stack  */
BYTE szBuf[60]; /* Will hold zero-terminated string returned by
wsprintf() below */
cbFileNameSize = 30; /* Size of module-name array */

IG_load_file ("picture.bmp", &hImage ); /* Try to load image */
nErrcount = IG_error_check(); /* Get number of errors on stack
*/
for ( i = 0;  i < nErrcount;  i++ )  /* For each error on stack:
*/
   {
   /* Get Module Name, Line Number, and Error Code:
*
   IG_error_get ( i, (LPSTR) &szFileName, cbFileNameSize,
           &iLineNumber, (LPAT_ERRCODE) &iCode, NULL, NULL );
/* Format error message in szBuf: */
   wsprintf ( szBuf, "Error %d in Module %s at Line %d", iCode,
              szFileName, iLineNumber );
/* Display error message in a Message Box, with heading
"Error":*/
   MessageBox ( NULL, szBuf, "Error", MB_OK );
   }
IG_error_clear();/* Done getting errors, clear the error stack
*/
```

This example displays the error number (`iCode`) in the Message Box. Text describing the nature of the error, and the ImageGear Error Name (such as `IGE_OUT_OF_MEMORY`) can be found in file `GEARERRS.H`.

It should be noted that each ImageGear function, excluding all `IG_dspl_...()` functions, clears the error stack upon entry. Therefore, after an ImageGear function call you should check the stack prior to your next ImageGear function call. Also, many ImageGear functions return as their value the value of **`IG_error_check()`**. So it is usually not necessary to call `IG_error_check()`. For example, the load-file and error-check statements in the above example could have been combined into one:

```
nErrcount = IG_load_file ( "picture.bmp", &hImage );
/* nErrcount now = number of errors place on stack during load
attempt */
for ( i = 0;  i < nErrcount;  i++ )
    …
```

Also there is a bunch of ImageGear functions that may help you with the error stack processing:

`IG_err_callback_get`

`IG_err_callback_set()`

IG_err_count_get()

IG_err_error_check()

IG_err_error_set()

IG_err_error_get()

IG_err_record_get()

IG_err_stack_clear()

Checking the ImageGear Version

ImageGear provides three IG_version_ …() functions. The first of these functions gives you update information:

```
INT nVersionMajor;
INT nVersionMinor;
INT nVersionUpdate;
IG_version_numbers ( &nVersionMajor, &nVersionMinor,
&nVersionUpdate);
```

The second IG_version_ …() function allows you to obtain the compilation date of the ImageGear that you are using:

```
LPSTR lpszCompileDate;
lpszCompileDate = IG_version_compile_date ( void );
```

Upon return from the above, lpszCompileDate will contain a pointer to a string of the form “Mmm dd yyyy,” such as “May 09 2001.”

Working with ImageGear Callback Functions

ImageGear’s strategy of providing callback function support for load, save, print, and other operations is to permit your application to help control these processes. Callback functions are functions that *you* write the code for, and whose names *you* provide to ImageGear. ImageGear will call them at appropriate breakpoints in an operation (such as after each raster line has been processed), at which time your function may modify image data, display status information, or perform other auxiliary operations you wish before returning control to ImageGear.

This section discusses how to declare, code, and invoke an ImageGear callback function; how to register an ImageGear callback function; and how to work with the status bar and tag callback functions.

Callbacks are actually function types (or templates) where you can include your own code to carry out extra operations during normal ImageGear file processing. Callbacks can be passed data, return data, or both. All callback type names begin with the prefix “LPFNIG” which stands for “**L**ong **P**ointer to a **F**u**N**ction of **I**mage**G**ear”. Due to this unusual prefix, their descriptions can easily be found in the section Core Component Callback Functions Reference in Chapter "Core Component API Function Reference".

Some callback functions return a Boolean value to ImageGear, indicating whether you want ImageGear to continue the operation, disregard the instructions in the callback, or abort an operation. Most callbacks are VOID, exchanging their information through their arguments.

How a callback function is coded, declared to ImageGear, and invoked by ImageGear, is illustrated by the simple examples below in which an application calls function **IG_load_file_display()** to load and then display an image. IG_load_file_display() will automatically call your callback of type **LPFNIG_LOAD_DISP**.

The following example shows a call to register a display callback and the callback itself:

```
VOID ACCUAPI  my_set_attributes_func
  (
     LPVOID  lpPrivateData, /* Ptr to private data area     */
     HIGEAR  hIGear         /* Handle of loaded image       */
   )
{
/* This callback function disables centering of the image: */
IG_dspl_layout_set( hIGear, IG_GRP_DEFAULT, IG_DSPL_ALIGN_MODE,
NULL, NULL, >NULL, 0,
IG_DSPL_ALIGN_X_LEFT|IG_DSPL_ALIGN_Y_TOP, 0, 0.0 );
return;
}
```

In response to the above call ImageGear loads the image, creates a DIB and a HIGEAR data structure, and then calls your callback function. When your callback function returns, ImageGear will display the image, and then return to the statement following **IG_load_file_display()**.

Note that your module containing the IG_load_file_display() should contain in its initial definitions a function prototype or declaration for the callback function. There are two ways that the callback can be declared:

1. By prototype:
   ```
   VOID ACCUAPI  my_set_attributes_func ( LPVOID lpPrivate, HIGEAR
   hIGear );
   ```
2. By declaration:
   ```
   LPFNIG_LOAD_DISP my_set_attributes_func;
   ```

NOTE: If you do not write code for a callback function type that is part of a normal API call, such as IG_load_file_display(), you can just pass in a NULL for the callback parameter.

Registering a Callback Function

You inform ImageGear of each callback function that you want it to call by specifying a pointer to that function as an argument in a call to an IG_...() function. The ImageGear functions that include at least one callback are:

- **IG_load_file_display()**
- **IG_file_IO_register()**
- **IG_load_tag_CB_register()**
- **IG_mem_CB_register()**
- **IG_save_tag_CB_register()**

- **IG_status_bar_CB_register()**

Some callbacks are registered with special registration calls while others are passed in as arguments to normal API calls. The reason for this is that some callbacks need to be called during more than one ImageGear process. Callbacks of this nature will generally be registered with a special registering function (which includes the word “register”) and called by ImageGear behind the scenes.

NOTE: If you’ve registered a callback function using an `IG_ …_CB_register()` function, you can un-register it by calling the function again, and supplying a `NULL` in place of the pointer to the callback function.

In each case, the `IG_…()` function’s description in "Core Component API Function Reference" chapter describes what type the callback function must be. This refers to the argument sequence with which ImageGear is going to call that particular callback function.

Status Bar Callback

ImageGear function **`IG_status_bar_CB_register()`** registers a callback function that will thereafter automatically be called by many ImageGear functions at the end of processing each raster line. The arguments let your callback function compute the completed percentage, so you can maintain and update a status bar or a message box showing this information.

Most of the following ImageGear functions will automatically call (not “register”) a status bar callback function if you’ve registered it:

- `IG_load_ …()`
- `IG_save_ …()`
- `IG_dspl_ …()`
- `IG_IP_ …()`
- `IG_FX_ …()`

To register your own status bar callback function (and the area you use if you want to pass your own data to it) make the following call:

```
IG_status_bar_CB_register ( my_status_bar_CB_func,
&myPrivateData );
```

Your status bar callback function must be of type **`LPFNIG_STATUS_BAR`**. Whenever called by ImageGear, your status bar callback function will be called with the following argument list:

```
BOOL ACCUAPI my_status_bar_CB_func ( LPVOID lpPrivate, PIXPOS
cYPos,
DIMENSION dwHeight );
```

Note the following:

- The function returns an `AT_BOOL` value, `TRUE` or `FALSE`. Return `TRUE` to have ImageGear proceed normally, return `FALSE` to tell ImageGear to put an `IGE_INTERRUPTED_BY_USER` error in the error stack and return from the `IG_…()` call it is processing (that is, to abort the operation returning the

above-named error).

- The second argument indicates the raster line number just processed.
- The third argument indicates the total number of raster lines in the image. However, since some functions do not process the lines in order, the quantity (`cYPos/dwHeight`) in general will not tell you the fraction completed. Instead, your callback function should count the number of times it has been called since the operation began, and divide this count by `dwHeight` to obtain the fraction completed.

Tag (Non-Image) Data Callback

NOTE: Through ImageGear preserves non-image callback functionality described in this section for the compatibility with applications based on previous versions of ImageGear, it also provides you with new callback functionality for non-image data processing. This new callbacks are connected with ImageGear Format Filters API. So if it is not imperative for you to use old callbacks, please refer to the new ones in the section “Non-Image Data Processing” on page -137 above in this chapter.

Firstly, for those of you who may not have been close-up with a TIFF file before, we offer an explanation of the word “tag”. While tags are not exclusive to the TIFF format, the word “tag” was borrowed from TIFF, which is an acronym that stands for “Tagged Image File Format”. Each TIFF image can have a set of “tags” totaling into the thousands. In other formats, the equivalents for a tag are normally referred to as a “fields”, “header information”, or “non-image data”. Essentially, tags constitute information that is stored in an image file in addition to the actual bitmap (image) data. Examples of such information range from the bare essentials, such as image width and height, to more auxiliary types of information, such as name of the artist or the date the image was created.

Throughout this section you will see the word tag used in a few different ways. To avoid confusion, here are the five types of tags we will be referring to:

- **Registered TIFF tags**: Those tags that are defined by the TIFF specification.
- **Reg istered tags**: This generic term refers to those tags that are defined by any format's specification. This will allow you to differentiate from "user-defined" tags.
- **User-defined tags**: Tags that you wish to define and add to a file when saving it with ImageGear. These tags are for your own use. If you are adding user-defined tags to a TIFF image, you must use a tag number of 32768 or higher so that you are not accidentally trying to overwrite "registered TIFF tags." As of this writing, TIFF is the only supported format that allows the storage of user-defined tags.
- **Volatile tags**: ImageGear refers to tags as "volatile" when the changing of their information could result in the corruption of the bitmap. ImageGear will not allow you to change the information in a volatile tag. Only ImageGear can change the data in these fields. For example, if you resize the image while it is

loaded, when you save it, ImageGear will update the height and width tags.

- **Non-volatile tags**: ImageGear refers to tags as "non-volatile" when the changing of their information cannot corrupt the image. Examples of such tags are the "Artist" tag (who created the image), and the "Date" tag (when the image was created). ImageGear allows you writing to non-volatile tags when you are saving the image.

ImageGear offers three callback function types for handling tag data. The two callback types **LPFNIG_TAG_GET** and **LPFNIG_TAG_SET** are used for *setting* and *getting* (respectively) the tag data of any of ImageGear's supported formats. The third, **LPFNIG_TAG_USER_GET** is used for *setting* private tag data. This means that you can create your own tags (or fields) and add them to the current set of tags.

NOTE: ImageGear's callbacks for getting and setting information are named from the toolkit's point of view. Since ImageGear is calling your callback functions, when the word “GET” is part of the callback name, it means that ImageGear is Getting information from you. When “SET” is part of a callback name, it means that ImageGear is Setting or sending information to you. If you can remember to use the toolkit’s point of view when using callbacks, the names will be easier to remember.

Reading Tags During Loading

When you load a file using ImageGear, you have the opportunity to read its tags. Before ImageGear loads any file it checks for the registration of a callback of type **LPFNIG_TAG_SET.** If such a callback has been registered, it will be called once for each tag that is being parsed. This includes both registered tags and user-defined tags. However, ImageGear will not let you write data to any tag while loading. When loading, tags are read-only. In your callback, you may wish to make copies of important data. Be sure to make a copy of the data, not the pointer to the data.

NOTE: For all non-volatile fields, ImageGear will store “default values”. To keep original data in such fields, you should make a copy of them while loading the image using a callback of type LPFNIG_TAG_SET function. Later, when saving the file, you can pass this data to a callback of type **LPFNIG_TAG_GET**. Your callback can be designed to look for certain types of tags and to make decisions based on the information in those tags, including whether you would want to modify the data in any of them during saving.

To register LPFNIG_TAG_SET call **IG_load_tag_CB_register().**

The example below shows a function of type LPFNIG_TAG_SET that will make a copy of the Artist tag of the image (TIFF) being loaded.

```
LPFNIG_TAG_SET send_me_tag_info; /* Declare my callback function
*/
HIGEAR hIGear; /* Will hold handle of image */
```

```
AT_ERRCOUNT  nErrcount; /* Returned count of errors */
static DWORD dwTags; /* Returned tag values */
nErrcount=IG_load_tag_CB_register ( send_me_tag_info, (LPVOID)
&dwTags );
nErrcount=IG_load_file("TaggedImage.TIF", &hIGear);
```

In response to the above call ImageGear loads the image, creating its DIB and HIGEAR data structure and handle, then calls your callback function once for each tag being parsed. Here is a simple example of what your callback function might look like:

```
BOOL ACCUAPI  send_me_tag_info
(
const LPVOID lpPrivate, /* Ptr to private data area */
AT_MODE nIGTag, /* Handle of loaded image */
AT_MODE nDataType, /* Data type of tag data */
const LPVOID lpTagData, /* a copy of the tag data */
DWORD dwSize /* size of the data */
)
{
/* This callback function makes a copy of the Artist tag */
LPBYTE lpDataPtr;
CHAR szArtistName [60];
switch (nIGTag){
…
case IGTAG_TIF_ARTIST:
   memcpy(&szArtistName, lpDataPtr, sizeof(szArtistName));
   return (TRUE);
   break;
return TRUE;
}
```

Modifying Tags During Saving

When you save any file using ImageGear, you have the opportunity to write to either non-volatile registered tags, or in the case of TIFF, to add user-defined tags. Before ImageGear begins saving your file it will look to see if either of the two tag save callbacks are registered: **LPFNIG_TAG_GET** or **LPFNIG_TAG_USER_GET**. If you have registered a callback of type LPFNIG_TAG_GET, ImageGear will check your callback each time it processes a non-volatile tag to see if you would like to overwrite the data in the tag. If you set the callback to TRUE, ImageGear will use your data to overwrite the value in this Tag; if you set the callback to FALSE, ImageGear will ignore your data for this particular tag. In your callback you might include a switch statement that checks the value of AT_MODE nIGTag.

To find out which TIFF tags are considered non-volatile, see the TIFF tag constants section of Geartags.h. The TIFF tag constants are accompanied by a key that tells you which tags will be written by ImageGear to the file when it is saved, which tags are sent to the user via callback, which tags are writable, and which ones can be modified by the user. Basically, if you find a “Y” (for yes) in columns 4 and 5, ImageGear will let you write to this tag. See Geartags.h and also see the note under the description for TIFF in File Format Reference Chapter.

In the example below both callbacks are registered by calling **IG_save_tag_CB_register()**:

```
LPFNIG_TAG_GET store_my_tag_info; /* Declare tag-modifying
callback*/
LPFNIG_TAG_USER_GET add_my_own_tags;/* Declare user-defined tag
adding*/
HIGEAR hIGear; /* Will hold handle of image */
AT_ERRCOUNT nErrcount; /* Returned count of errors */
static DWORD dwTags; /* Returned tag values */
nErrcount = IG_save_tag_cb_register ( store_my_tag_info,
add_my_own_tags, (LPVOID) &dwTags );
nErrcount = IG_save_file(hIGear, "TaggedImage.TIF",
IG_SAVE_TIF_JPG);
```

In response to the above call ImageGear begins to save the image pointed to by `hIGear` as a JPEG-compressed TIFF. As it reaches each non-volatile writable tag it will call your function `store_my_tag_info()`. Note, that it also registers `add_my_own_tags()` which will be called after `store_my_tag_info()` to add any desired user-defined tags. The example below writes to the Artist tag when the file is being saved.

```
BOOL ACCUAPI store_my_tag_info;
(
LPVOID lpPrivate,/* Private data passed in */
AT_MODE nIGTAG,/* Tag ID value */
LPAT_MODE lpDataType, /* type of data being set */
LPVOID lpData, /* pointer to the data */
DWORD dwDataLen /* size of data in bytes */
)
{
/* This callback function modifies the Artist tag */
LPPRIVDATASTRUCT lpStructPtr;
lpStructPtr =((LPPRIVDATASTRUCT) lpPrivate);
   if (lpStructPtr->PublSetUsd){
      switch (nIGTag){

      case IGTAG_TIF_ARTIST:
      *lpDataType = IG_TAG_TYPE_ASCII;
         memset( (LPBYTE)(lpData), '\0', dwDataLen );
         memcpy((LPBYTE)(lpData),"JOE PROGRAMMER",14);
         return (TRUE);
         break;
      default:
         return (FALSE);
         break;
      }
   }
return (FALSE);
};
```

NOTE: When you set a tag you have to tell ImageGear the type and size of your data. When your callback is called, ImageGear will pass you the allowable data size for the tag. You must be sure not to exceed this length.

ImageGear Default Tag Values

For those TIFF tags that can be written ImageGear uses default values unless you tell it otherwise. In other words, if you do not register a callback function of type **LPFNIG_TAG_USER_GET**, ImageGear will set all writable TIFF tags to ImageGear default values. An example of one such tag is the Artist tag. Let's say it originally contains an artist name of “Giuseppe Artisano”. If you do not provide data for this tag and return TRUE from your callback, ImageGear will modify it with its own default value of “1996-2001 AccuSoft Co., All rights reserved”. To keep the name of Giuseppe Artisano in the Artist Tag you need to make a copy of this string via your **LPFNIG_TAG_SET** function, and when saving the image, use a callback of type **LPFNIG_TAG_USER_GET** to set the tag data back to its original value.

User-Defined Tags

As mentioned earlier, the other type of tag that you can write is a user-defined tag. These are tags that you register for your own use. To add user-defined tags to an image that is being saved, register a callback of type **LPFNIG_TAG_USER_GET**.

Until you return a FALSE from your callback it will continue to call it and add your list of tags to the image being saved. To register a callback of type LPFNIG_TAG_USER_GET call **IG_save_tag_CB_register()**. Remember that if you are saving to a TIFF file, your private tags must have values greater than 32768.

For more information about tag callbacks please see the individual descriptions for the tag callbacks prototypes in Core Component API Function Reference. The sample application, Tifsamp.c demonstrates all three tag callbacks discussed in this section.

NOTE: Remember, that as of this writing, TIFF is the only file format which supports the appending of user-defined tags.

Working with Stripped Images

Large images often test the patience of computer users when they are loaded or displayed. To help speed up loading and display times ImageGear provides alternate storage schemes that allow a large image to be loaded in segments. Such segments could be loaded and pieced together in sequential order.

One of the storage schemes developed was stripped storage. A stripped image contains offsets to numerous groups of pixel rows. Each strip is the full width of the image. The diagrams below show an image stored in strips (shown separated), and the same image pieced together in memory:





In the actual DIB there are no spaces between strips. The header simply contains the address of the leftmost pixel of the first row of each strip. The spaces between the strips in the image on the left are just for illustrative purposes. On the right, the image has been loaded and pieced together in memory.

The TIFF format, in which the use of stripped storage was once commonplace, rarely uses this scheme today. However, there are still many old images still in use that were created with this storage scheme, and ImageGear fully supports these images.

The loading and saving of stripped images using ImageGear is no different than loading or saving a non-stripped image. You can use **`IG_load_file()`** or any other ImageGear loading API, without making any special settings. When loading, ImageGear will automatically detect the stripped storage and piece all of the strips together in the proper order.

Once in memory, a stripped image will be like any other image loaded into memory. It will be a DIB with a `HIGEAR` handle. You can now save it to any ImageGear-supported format. If you would like to save it to again using stripped storage, you may do so using ImageGear Format Filter **`IG_fltr_ctrl_set()`**/ **`IG_fltr_ctrl_get()`** functions. Of course, you must choose a file format that supports stripped storage. You can save a TIFF image with stripped storage by setting the number of strips to use. Here is the call you would make:

```
/* get current value of NUMBER_OF_STRIPS control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "NUMBER_OF_STRIPS", FALSE,
NULL, NULL, (LPVOID)nNumberOfStrips, sizeof(nNumberOfStrips));

/* set new value to NUMBER_OF_STRIPS control parameter */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "NUMBER_OF_STRIPS", (LPVOID)3,
sizeof(nNumberOfStrips));
```

In fact, there are a number of ways that a TIFF may be stored using **`IG_fltr_ctrl_set()`** with `WRITE_CONFIG` parameter. See this control parameter and its range of settings in TIFF Control Parameters Table located in the section ImageGear Supported File Formats Reference.

TIFF filter control parameters that pertain to stripped storage include those shown in the following list:

- `NUMBER_OF_STRIPS`
- `BUFFER_SIZE`
- `WRITE_CONFIG`

Please see the section TIFF in File Format Reference Chapter for more detailed information about every of these parameters.

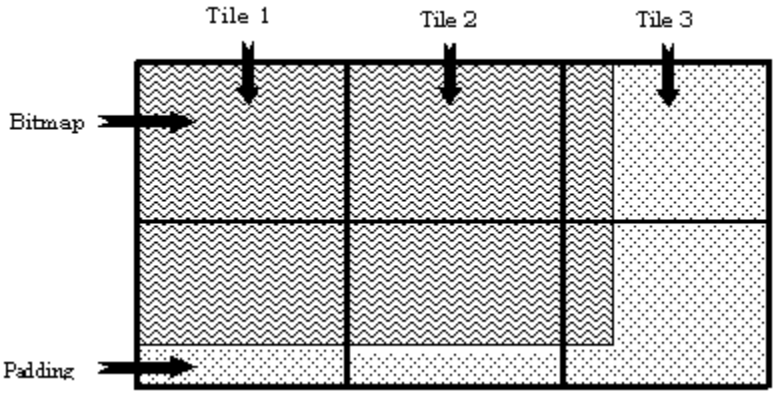
Working with Tiled Images

In a tiled image, pixel data is stored in blocks called “tiles”, whose height and width are less than that of the height and width of the full image. All tiles are the same size. However, it is rare that the tiles evenly cover the image. Therefore, in most cases, the following statements will be TRUE:

```
(# of tiles per row) × (width of a tile)  ≠  the width of the
image
(# of tiles per column) × (height of a tile)  ≠  the height of
the image
```

Padding

Those areas of tiles that are on the border of an image, but are not completely filled with image data, are filled with padding. The image below illustrates a tiled image and shows padding to the right side and bottom of the image. Padding to the right and bottom of the image is the most frequent tiled storage situation that you will encounter:



When ImageGear loads a tiled image (or any image) it reads the header or tag data to find out the width and height of the actual image. This data will not include the padding. As ImageGear loads the file, it will discard the padding. The following ImageGear-supported file formats allow the use of tiled storage: TIFF, IBM IOCA, and IBM MO:DCA. As of this writing, the highest level of control given to you for working with tiled images is for the popular TIFF format, which will be mentioned frequently throughout this section.

If you were to call **`IG_load_file()`** to load a tiled image, it would load the first tile of the first page. If you want to load more than one tile and stitch them together, you should call one of the `IG_load_tiles_stitch …()` functions. In addition to loading and stitching any number of tiles, ImageGear also allows you to query the tiles of an image before loading it, tell ImageGear which tiles to “stitch together” when loading, and how to tile the image when you save it to disk.

If you wish to specify which tiles of a TIFF image to load and stitch, your first task will be to find out whether the image actually has tiles or not. The function **IG_tile_count_get()** can be called to get this information. This function returns the number of tiles per row, and the number of tiles per column. If this function returns zeros for the number of tiles across the image (lpTileCols) and the number of tiles down the length of the image (lpTileRows), you will know that the image is not tiled. There are two other versions of this function: if you have already opened a TIFF file and have a File Descriptor handle for it, use the function **IG_tile_count_get_FD()**; if the file has already been loaded into memory (using **IG_load_mem()**), use the function **IG_tile_count_get_mem()**.

The following ImageGear functions can be used to load and stitch TIFF tiles:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch(const LPSTR
lpszFileName, UINT nPage, LPAT_STITCH lpStitch, LPHIGEAR
lphIGear);

AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_FD(INT fd, LONG
lOffset, UINT nPage, LPAT_STITCH lpStitch, LPHIGEAR lphIGear);

AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_mem(LPVOID lpImage,
DWORD dwImageSize, UINT nPage, LPAT_STITCH lpStitch, LPHIGEAR
lphIGear);
```

Each function takes a page number specifying which page of a multi-page file to load. Set this to 1 if it is not a multi-page file. Each also takes a structure of type AT_STITCH that will tell ImageGear which tiles to stitch together and load. Here is the definition of AT_STITCH:

```
typedef struct tagAT_STITCH
{
LONG uRefTile; /* Upper left-hand corner tile # */
LONG uTileRows; /* Number of tiles to stitch across*/
LONG uTileCols; /* Number of tiles to stitch down */
}AT_STITCH, FAR *LPAT_STITCH;
```

Set the structure member uRefTile to the number of the tile that you would like to be used as the upper left-most tile in the image that will be loaded and stitched. Set uTileRows to the number of tiles across that you would like your stitched image to use. Set uTileCols to the number of columns of tiles that you would like your stitched image to use. If you wish to stitch all tiles together, simply pass in a NULL for this structure.

Below are some examples of how a tiled image can be stitched together. **Figure 1** shows a tiled TIFF image that contains a total of 9 tiles. Note that for simplicity, the tiles are shown to evenly cover the exact height and width of the image. This rarely happens.

The numbering scheme in **Figure 1** reflects the way that ImageGear keeps track of the tiles. When you refer to specific tiles, the upper-left-most tile will be 1. The tiles are then numbered sequentially from left to right, top to bottom.

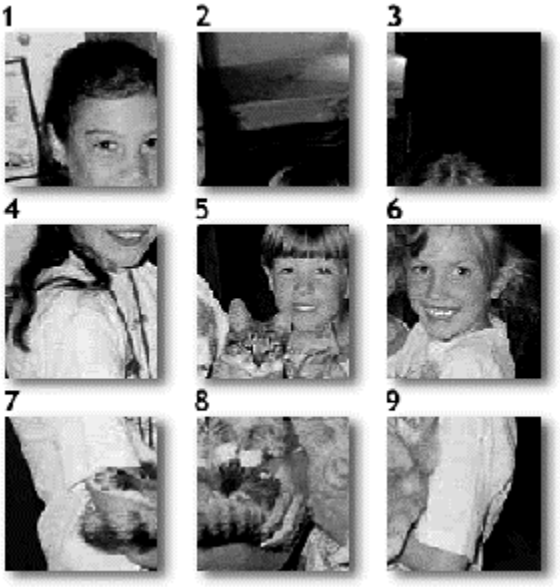


Figure 1. TIFF image with 9 tiles. The numbering scheme shown is the same one you should use when interfacing with tiles using ImageGear.

Figure 2 shows all of the tiles stitched together, which would be the result if you set the `AT_STITCH` structure to `NULL`.



Figure 2. All of the tiles loaded and stitched together.

The following call achieves the results shown in **Figure 2**:

```
IG_load_tiles_stitch(“KidsKatz.tif”, 1, NULL, &hIGear)
```



Figure 3. Tiles 5, 6, 8, and 9 loaded and stitched together.

To achieve the results shown in **Figure 3**, make the following calls:

```
AT_STITCH stitchStruct;
HIGEAR hIGear;
stitchStruct.uRefTile = 5;
stitchStruct.uTileRows = 2;
stitchStruct.uTileCols = 2;

IG_load_tiles_stitch(“KidsKatz.tif”, 1, &stitchStruct, &hIGear)
```

You may want to find out the width and height of the tiles in the image so that you can make an informed decision about which tiles to load. First use the function **`IG_info_get_ex()`** which will return you the width and height of the image. Then divide the width of the image by the number of tiles per row, and the height of the image by the number of tiles per column. To get the height and width of an image which is already opened and for which you have a File Descriptor handle, call **`IG_info_get_FD_ex()`**; to get the height and width of an image which has been loaded into memory and for which you have a `HIGEAR` handle call **`IG_info_get_mem_ex()`**.

Automatic Tile Stitching

ImageGear allows you to automatically stitch all image tiles during loading. Set “STITCH_TILES” filter control parameter to TRUE to enable automatic tile stitching. In this mode:

- Image loading functions load the whole image as if it was not tiled;
- Tile counting functions report that the image is not tiled (has a single tile);
- Header reading functions report full image dimensions rather than dimensions of a single tile.

Set “STITCH_TILES” filter control parameter to FALSE to disable automatic tile stitching. In this mode:

- Image loading functions load a single tile (by default - first image tile);
- Tile counting functions report actual number of tiles;
- Header reading functions report dimensions of a single tile.

Currently, the following ImageGear format filters support automatic tile stitching:

- IBM AFP

- IBM IOCA
- IBM MO:DCA
- JPEG 2000
- JPX
- MJ2
- TIFF

Saving a TIFF File Using Tiles

When you save a TIFF image to disk, you can tell ImageGear to save the bitmap data as tiles, regardless of what storage scheme was originally used. To choose tiled storage and to make modifications to the way ImageGear will implement these storage schemes, use the **`IG_fltr_ctrl_set()`** and **`IG_fltr_ctrl_get()`**functions. For more information on getting/setting filter control parameters see the section “Using Format Filters API for Filter Control” on page -135.

There the list of TIFF filter control parameters that are related to tiles:

- `WRITE_CONFIG`
- `TILE_H_COUNT`
- `TILE_V_COUNT`
- `TILE_WIDTH`
- `TILE_HEIGHT`
- `BUFFER_SIZE`

Please see the section TIFF in File Format Reference Chapter for more detailed information about every of these parameters.

The following example call shows an example image loaded into memory with just tiles 5, 6, 8, and 9 and then being saved as a tiled image with 20 tiles:

```
/* Declare a structure of type AT_STITCH */
AT_STITCH stitchStruct;
UINT nPage;
HIGEAR hIGear;
LONG lHorizTileCount, lVertTileCount;

/* Set uRefTile to the number of the tile that you would like to
use as the upper-left most tile in the stitched image */
stitchStruct.uRefTile = 5;

/* Set uTileRows to the number of tiles that you would like per
row */
stitchStruct.uTileRows = 2;

/* Set uTileCols to the number of tiles that you would like per
column */
stitchStruct.uTileCols = 2;
```

```
/* Set lHorizontalTileCount to the number of tiles per row that
ImageGear should make when an image is saved to disk as a tiled
TIFF. */
lHorizTileCount = 5;

/* Set lVertTileCount to the number of rows that ImageGear
should make when an image is saved to disk as a tiled TIFF. */
lVertTileCount = 4;

/* Set nPage to the page number of a multipage TIFF that you
would like to load. If the TIFF isn't multipage, set to 1. */
nPage = 1;


/* This function loads the tiles specified by &stitchStruct,
stitches them together in a DIB and returns you a HIGEAR to the
image in &lphIGear.*/
IG_load_tiles_stitch("KidsKatz.tif", nPage, &stitchStruct,
&hIGear);

/* Use the Filter Control function to tell ImageGear how to
configure TIFF files when it saves them to disk. This call
specifies that they should be saved as tiled images and that you
will be providing ImageGear will a fixed number of tiles to
create when saving the file. */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "WRITE_CONFIG",
(LPVOID)IG_TIF_TILED_FIXED_COUNT, sizeof(lpWriteConfig));

/* Use the Filter Control function to tell ImageGear how many
tiles per row you would like to make in images saved to disk as
TIFFs. */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "TILE_H_COUNT",
(LPVOID)lHorizTileCount, sizeof(lHorizTileCount));

/* Use the Filter Control function to tell ImageGear how many
rows of tiles you would like to make in images saved to disk as
TIFFs. */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "TILE_V_COUNT",
(LPVOID)lVertTileCount, sizeof(lVertTileCount));

/* Call IG_fltr_save_file() to save the current HIGEAR image as
an uncompressed TIFF. */
IG_fltr_save_file(hIGear, "KidsKatz.tif", IG_SAVE_TIF_UNCOMP,
1, TRUE);
```

TIP: If you do not specify the configuration to use for saving a TIFF to disk, ImageGear will use the default setting of `IG_TIF_STRIP_FIXED_COUNT`. If you do not specify a number of strips, the default of 1 is used.

TIFF Storage

TIFF format filter supports four modes of storage for writing out a TIFF file. All settings are made by supplying "WRITE_CONFIG" control parameter for function **IG_fltr_ctrl_set()**. Here are the possible settings:

IG_TIF_STRIP_FIXED_COUNT	Write the image using a fixed number of strips. The number of strips to use can be set via the NUMBER_OF_STRIPS control parameter.
IG_TIF_STRIP_FIXED_BUFFER	Write the image using strips so that each strip is not greater than the specified size in bytes. The size of the strip buffer can be set via the BUFFER_SIZE control parameter. Please note that at least one raster will be included in the strip.
IG_TIF_TILED_FIXED_SIZE	Save the image using tiles of fixed size. The size of the tiles can be set via the TILE_WIDTH and TILE_HEIGHT control parameters.
IG_TIF_TILED_FIXED_COUNT	Save the image using a fixed number of tiles. The number of tiles in both the horizontal and vertical direction can be set via the control parameters TILE_H_COUNT and TILE_V_COUNT.

Working with Run Ends Format

As of ImageGear v14.5, the Windows DIB format is no longer used for internal storage of images. So 1-bit images are now always stored internally in run ends format. Previous versions of ImageGear had IG_IP_convert_runs_to_DIB() and IG_IP_convert_DIB_to_runs() functions which converted the internally stored image between run ends and DIB (uncompressed packed) format. These functions are no longer necessary because conversion is performed automatically as needed.

When you read an image's pixel data using a pixel access function such as **IG_DIB_raster_get()**, ImageGear will decompress the pixel data for that raster and store it in your buffer. You can specify packed (8 pixels per byte) or unpacked (1 pixel per byte) format. When you write pixel data using a pixel access function such as **IG_DIB_raster_set()**, ImageGear will compress and store the pixel data in run ends format. It's important to realize that this decompression and compression is the same work that was previously performed in IG_IP_convert_runs_to_DIB() and IG_IP_convert_DIB_to_runs(). But instead of being performed on the entire image before and after processing, this work is performed on parts of the image during processing.

Decompressing and Compressing the Entire Image

If you want to work with an image directly and to avoid using pixel access functions on a per-raster basis, you can decompress the image to your own buffer using **IG_DIB_area_get()**. Here's an example scenario:

1. Call **IG_DIB_area_size_get** to get the size of the buffer.
2. Allocate the buffer (i.e. with new or malloc).
3. Call **IG_DIB_area_get**. ImageGear decompresses the image into your buffer.
4. Read/write uncompressed pixel data directly in your buffer.
5. Call **IG_DIB_area_set** if updating ImageGear's internal copy of the image is desired. ImageGear compresses the image from your buffer.

Run Ends Format Description

The run ends format is a specialized variant of run length encoding. Run length encoding relies on the fact that certain types of images frequently contain parts where many adjacent pixels share the same color. A description of such an occurrence is known as a run. Typically a run is described as 1) a color and 2) the number of following pixels that are that color. An image raster (or entire image) can be stored as a collection of runs. For example, an image of this page could be described as "2000 white pixels, 5 black pixels, 15 white pixels, 5 black pixels, 15 white pixels, 5 black pixels, 30 white pixels" and so on.

Since the run ends format only works on 1-bit images, it can take advantage of the fact that there are only two possible colors present in the raster: 0 and 1. Since there are only two possible colors, the color does not need to be stored for each run. It is inferred from the previous run. Also, having only two colors makes it especially likely that long runs of identically colored pixels will occur, as compared to images with more colors present.

The following points characterize the run ends format:

- An image is stored as a collection of rasters encoded in run ends format. Each raster is independent – there is no information shared between rasters. Therefore, consider only a single raster when thinking about the run ends format.
- A run ends raster is stored as an array of run ends. A run end is a value of type AT_RUN which marks the end of a run by storing the horizontal position (X-coordinate) of where the next run begins.
- Run ends are stored in order from left to right.
- It is always assumed that the first run in a raster is white. If it is not, there will be a "null run" at the beginning of the raster which ends at column 0. This is a means of getting the first real (non-zero-length) run to be black.
- The last run end in the raster is always equal to the image width. This value is stored three times to mark the end of the raster.

Here are some examples of rasters that are 8 pixels in width. Each raster is shown first in uncompressed format, then in run ends format as it would be stored in memory on a 32-bit x86 platform. That is, the number 5 is stored in memory as "05 00 00 00".

Example 1	Example 2
11001000 02 00 00 00 // white run until column 2 04 00 00 00 // black run until column 4 05 00 00 00 // white run until column 5 08 00 00 00 // done (remainder is black) 08 00 00 00 08 00 00 00	00000101 00 00 00 00 // *get first run to be black* 05 00 00 00 // black run until column 5 06 00 00 00 // white run until column 6 07 00 00 00 // black run until column 7 08 00 00 00 // done (remainder is white) 08 00 00 00 08 00 00 00

Accessing Run Ends Data with IG_runs_row_get/set

You can read and write rows of run ends data by using the `IG_runs_row_get/set` functions. **`IG_runs_row_get()`** retrieves a pointer to the run ends data. **`IG_runs_row_set()`** updates a row with compressed data from a buffer you supply. These functions are the recommended way of accessing run ends data. The format of the data is exactly as described in the previous section.

Accessing Run Ends Data with IG_image_DIB_raster_pntr_get

`IG_image_DIB_raster_pntr_get()` is a general purpose function for getting a pointer to pixel data for a given raster. If you use it on a 1bpp image, it will return a pointer to a run ends raster. You can access this raster directly, but be aware of the following:

- There is an additional `AT_RUN` value at the beginning of the raster. This value is equal to the total number of `AT_RUN` values used to store the raster, including this value. For example, for the raster “11001000”, this value would be 7.
- You cannot write data which exceeds the original length of a raster, because ImageGear allocates only enough space to hold the runs for that raster. For this reason, it is safer to use **`IG_runs_row_set()`** which can reallocate if necessary.
- Run ends rasters are not stored contiguously in memory. You must call **`IG_image_DIB_raster_pntr_get()`** for each raster you want to process.

Sample Run Ends Code

Here’s a sample function which decompresses a run ends raster into uncompressed unpacked (1 byte per pixel) format. It’s designed to work with the data you would get from **`IG_runs_row_get()`**.

```
// runsToUnpacked: Decompresses a run ends raster to unpacked
format.
//   nWidth – width of image in pixels
//   lpRuns – pointer to input buffer containing run ends data
//   lpPixels – pointer to output buffer to receive unpacked
pixel data
void runsToUnpacked(AT_DIMENSION nWidth, LPAT_RUN lpRuns,
LPAT_PIXEL lpPixels)
{
   // Starting color is white
   AT_PIXEL outputPixColor = 1;
   // Loop through runs
   AT_INT outputPixPos = 0;
   while (1)
   {
      // Find out when the current run ends
      AT_RUN runEnd = *lpRuns++;
      // Fill in pixels for this run
      while (outputPixPos < runEnd)
         lpPixels[outputPixPos++] = outputPixColor;
      // Have we reached the end?
      if (outputPixPos >= nWidth)
         break;
      // Switch colors for next run
      outputPixColor = !outputPixColor;
   }
}
```

Here's a more minimalist view of the same function:

```
void runsToUnpacked(AT_DIMENSION w, LPAT_RUN lpRuns, LPAT_PIXEL
lpPixels)
{
   AT_PIXEL c = 1;
   AT_INT x = 0;
   while (1)
   {
      AT_RUN r = *lpRuns++;
      while (x < r)
         lpPixels[x++] = c;
      if (x >= w)
         break;
      c = !c;
   }
}
```

Here's a more complex function that creates a 90-degree rotated copy of an image. It operates entirely on run ends data without ever decompressing the data. Note that this is only sample code. This does not represent how ImageGear works internally. Also, error handling is omitted.

```
// Returns a 90-degree rotated copy of the source image
HIGEAR rotate90(HIGEAR hImageSrc)
{
   HIGEAR hImageDst = NULL;
```

```
   HIGDIBINFO hDIB;
   AT_INT d[1] = { 1 };
   AT_DIMENSION srcWidth, srcHeight, dstWidth, dstHeight;
   AT_PIXPOS x, y;

   // Get info about source image
   IG_image_dimensions_get(hImageSrc, &srcWidth, &srcHeight,
NULL);

   // Create destination image
   dstWidth = srcHeight;
   dstHeight = srcWidth;
   IG_DIB_info_create(&hDIB, dstWidth, dstHeight,
IG_COLOR_SPACE_ID_I, 1, d);
   IG_DIB_palette_alloc(hDIB);
   IG_image_create(hDIB, &hImageDst);
IG_DIB_info_delete(hDIB);
   AT_RGB rgb = { 255, 255, 255 };
   IG_palette_entry_set(hImageDst, &rgb, 1);

   // Make a list of source raster pointers
   LPAT_RUN *lpSrcRasters = NULL;
   lpSrcRasters = (LPAT_RUN *) malloc(sizeof(LPAT_RUN) *
srcHeight);
   // Make a list of current colors for source runs
   LPAT_BYTE lpSrcRunColors = NULL;
   lpSrcRunColors = (LPAT_BYTE) malloc(sizeof(AT_BYTE) *
srcHeight);

   // Populate the lists
   for (y = 0; y < srcHeight; y++)
   {
      AT_RUN runCount;
      IG_runs_row_get(hImageSrc, y, &runCount,
&lpSrcRasters[y]);
      if (*lpSrcRasters[y])
         lpSrcRunColors[y] = 1;
      else
      {
         lpSrcRunColors[y] = 0;
         lpSrcRasters[y]++;
      }
   }

   // Allocate a raster large enough to store worst-case input
data
   LPAT_RUN lpDstRaster = (LPAT_RUN) malloc(sizeof(AT_RUN) *
(dstWidth + 4));

   // Loop through output rasters
   for (y = 0; y < dstHeight; y++)
   {
```

```
            AT_INT nDstRuns = 0;
            AT_BYTE dstRunColor = 1;
            AT_INT srcRasterIndex = srcHeight - 1;
            // If the first source pixel is black,
        // set us up to start with black in the output raster
            if (!lpSrcRunColors[srcRasterIndex])
            {
                dstRunColor = 0;
                lpDstRaster[nDstRuns++] = 0;
            }
            // Loop through columns in destination image
            for (x = 0; x < dstWidth; x++)
            {
                // Check the color of the run in the source raster
        that
                // corresponds to the current column in the
        destination raster.
                // Is it the same color as the run we're currently
        constructing?
                if (lpSrcRunColors[srcRasterIndex] != dstRunColor)
                {
                    // If not, then we need to store the run we've been
        making
        // in the destination raster.
                    lpDstRaster[nDstRuns++] = x;
                    // Alternate the current destination run color
                    dstRunColor = !dstRunColor;
                }
                // See if it's time to move on to the next source run
        for
        // this source raster
                if (*lpSrcRasters[srcRasterIndex] == y)
                {
                    lpSrcRasters[srcRasterIndex]++;
                    lpSrcRunColors[srcRasterIndex] =
        !lpSrcRunColors[srcRasterIndex];
                }
                // Move on to the next source raster (go *up* through
        the source image)
                srcRasterIndex--;
            }

            // Add the three ending runs to the destination raster
            lpDstRaster[nDstRuns++] = dstWidth;
            lpDstRaster[nDstRuns++] = dstWidth;
            lpDstRaster[nDstRuns++] = dstWidth;
            // Store the destination raster!
            IG_runs_row_set(hImageDst, y, nDstRuns, lpDstRaster);
        }

        // Clean up
        free(lpSrcRasters);
```

```
    free(lpSrcRunColors);
    free(lpDstRaster);

    return hImageDst;
}
```

Working with ImageGear Digital Camera RAW Filter

ImageGear supports reading raw images created by many digital cameras. Support for raw images includes reading of image pixel data, color reconstruction of raw images, and reading of metadata stored with the images.

File format

Since there are many flavors of raw formats, the list is continuously growing as new vendors/models enter the market, and the formats share many similarities, ImageGear treats all raw formats as one format: `IG_FORMAT_DCRAW`. Details regarding manufacturer, model, etc, are available as metadata.

Loading raw images

Load camera raw images in the same way as you load images of all other formats, using one of the image loading methods. Use control parameters of the `IG_FORMAT_DCRAW` filter to control loading of raw images.

By default, ImageGear attempts to reconstruct the full color image from raw data. You can also load unprocessed raw image, to apply some custom processing to it.

Loading of pixel data from raw and DNG images

Digital cameras store raw images in exactly the same format as obtained from the camera sensor. A number of image processing operations must be performed to reconstruct full color image from the raw data.

The most important operations are listed below.

- **Trimming of the black frame, subtraction of black level.** Black level, or Thermal Black is the level of signal produced by heating of the camera rather than by light. Many camera sensors measure the black encoding level using fully masked pixels at the edges of the sensor. ImageGear trims these pixels from the image during loading. Intensities of these pixels are averaged to produce the Black Level value. Black Level should be subtracted from image pixels for correct reconstruction of the image.
- **Intensity stretching according to White Level.** White level specifies the maximum intensity that can be recorded by camera sensor, or an intensity at which the sensor's output becomes highly nonlinear. Image intensities between the BlackLevel and WhiteLevel range should be stretched to the full contrast range supported by the image bit depth.

- **Demosaicing.** Most of digital cameras store pixels in "mosaic" format. At a given pixel location either a Red, Green, Blue, Cyan, or some other color sample value is recorded. Such images are referred to as "Color Filter Array" type. A two-dimensional matrix called "Color Filter Array pattern" describes positions of pixels of particular color in the mosaic image. In order to recreate the full color values in each pixel, it is necessary to interpolate neighbouring pixels.
- **White balance adjustment.** Most cameras store the information on white balance setting that was used at the time of image capture.
- **Color space conversion,** from camera color space to the working RGB space. At the moment, only sRGB is supported as destination color space.
- **Contrast and gamma adjustment.**
- **Image rotation.** Some camera sensors are rotated by 45 degrees. Such images need to be rotated back.

ImageGear has two modes for loading raw images:

1. **Color reconstruction mode (default).** In this mode, ImageGear attempts to reconstruct the full color image. This includes all steps outlined above.
2. **Raw mode.** In this mode, ImageGear loads pixels as they are stored in the image and does not apply any image processing operations, except for trimming of the black frame from the image. BlackLevel values are not subtracted from image pixels. Color filter array type images are loaded as 1-channel grayscale. The application can reconstruct the full color image, using the information taken from metadata.

Use `RECONSTRUCT_COLORS` control parameter of the `IG_FORMAT_DC_RAW` and `IG_FORMAT_DNG` filters to switch between these modes.

Note that camera manufacturers do not publish specifications on their raw formats. ImageGear attempts to reconstruct color images correctly where it is possible. For some cameras not all of the color balance data is available. The application may need to apply additional processing to such images.

The section ImageGear Supported Digital Camera RAW Formats lists the level of support that ImageGear provides for particular cameras.

Control parameters for the Camera Raw filter

Filter Control Parameter	Type	Default Value	Available Values	Description
RECONSTRUCT_COLORS	AT_BOOL	TRUE	TRUE, FALSE	ImageGear attempts to reconstruct full color image from the camera raw image. Otherwise, ImageGear loads raw pixel data without any processing.

Using ImageGear Component Manager

The general structure of ImageGear is a component structure that consists of the core (main) ImageGear module and some additional components.

The component is code that can be connected with the main ImageGear functionality using a platform-independent API. A component can be a separate module or linked statically with the main IG module. To initialize component functionality, attach (load) the component to the main ImageGear module through the special ImageGear component manager API. When the component is attached, it gets access to all core ImageGear functions, and the core ImageGear functionality can use the component's structures, functions, and control parameters.

Usually, ImageGear components contain such additional functionality as additional format filters (for example, LZW, PDF) and additional image processing functionality (such as ART).

Once you have loaded the component it cannot be unloaded up to the moment of cleanup of the main ImageGear module, i.e. exit of ImageGear based application.

ImageGear identifies every component by its name, and during the attaching process it calculates the physical name of the file where the component is located. By default ImageGear assumes that all components are located in the same directory where the main IG module is located, however there is a way to set up a full path to the module for every component.

See also the ImageGear Component Names section.

Component Manager API

ImageGear provides the component manager API that can be used for attaching, checking, function requesting, and retrieving information of every ImageGear component.

To attach (load) the component to the core ImageGear you can use the function:

```
IG_comm_comp_attach(LPCHAR lpCompName)
```

This function attaches the component, determined by `lpCompName` (for instance, "`CAD`"), to the core IG module. If a component with the given name is already attached, this function does nothing. See also the ImageGear Component Names section.

To check if a component with a given name is currently attached, use the function:

```
IG_comm_comp_check(LPCHAR lpCompName)
```

If the component is attached, then this function returns `TRUE`; it returns `FALSE` otherwise.

It is the usual situation that some component provides functionality that should be accessible from the application, and there are two ways that the application can call some function implemented by the component. The First method is using the component manager function:

```
IG_comm_function_call(LPCHAR lpEntryName, ... );
```

where lpEntryName is a name of the requested function in the form "<COMP_NAME>.<FUNC_NAME>", where <COMP_NAME> is a name of the component that provides the function and <FUNC_NAME> is a name of the function.

However, the application is not supposed to call this function directly, but use a special macro that is provided for each public function by the component. That macro is located in the special header file i_<COMP_NAME>.h named **component public header**. For instance, i_CLN.h (ImageClean component):

```
#define IG_IC_clean_borders_ex( hIGear, nLeftBorderSize,
nRightBorderSize, nTopBorderSize, nBottomBorderSize,
nMinLinesNum, nMinLineWidth)
    ((AT_ERRCOUNT (CACCUAPI *)(LPCHAR, HIGEAR, UINT, UINT, UINT,
UINT, UINT, UINT))
IG_comm_function_call)("CLN.IG_IC_clean_borders_ex",  hIGear,
nLeftBorderSize,  nRightBorderSize,  nTopBorderSize,
nBottomBorderSize,  nMinLinesNum,  nMinLineWidth)
```

Another way is to use the function

```
IG_comm_entry_request( LPCHAR lpEntryName, LPAFT_ANY
*lpFuncPtr, LPCHAR lpReason );
```

The first parameter is the name of the function in the format described above. The second parameter is a pointer to the necessary function. The last parameter is a text description reason to get access for this function (it is optional and can be NULL). The component public header usually contains a type declaration for any public function. The correct way to call such a function would be to declare the variable of the necessary function type defined in a public header; use IG_comm_entry_request() to initialize this variable with the correct value and then call it.

NOTE: To distribute your application using any ImageGear component API, you should include in your application's *.c file the component public header i_<COMP_NAME>.h.

Also, there is a function to retrieve information about all attached components:

```
IG_comm_comp_list(LPUINT *lpnCount, UINT nIndex, LPCHAR lpComp,
DWORD dwCompSize, LPUINT lpnRevMajor, LPUINT lpnRevMinor, LPUINT
lpnRevUpdate,
LPCHAR lpBuildDate, UINT nBDSize, LPCHAR lpInfoStr, UINT
nISSize)
```

This function provides you with the full list of info about the component determined by nIndex index from the list of currently attached components whose numbers are returned trough the lpnCount argument.

ImageGear Component Names

Below is the list of ImageGear components and their short names that you should provide for **IG_comm_comp_attach()** through the lpCompName argument:

- Core ImageGear Component - "CORE"
- ImageGear ART Component - "ART"

- ImageGear LZW Component - "LZW"
- ImageGear PDF Component - "PDF"

Using ImageGear ARTX Component

NOTE: You can only attach one ART component. ARTX cannot be attached when ART is already attached.

Using ImageGear ART Component

NOTE: You can only attach one ART component. ART cannot be attached when ARTX is already attached.

ImageGear ART Component helps developers to add redlining and annotation capabilities to any application built with ImageGear. If you have an application that can read, display, and print images, with ART component attached you can add marking capabilities, such as arrows, highlighting, and sticky notes to documents images, and use them as though you were working on printed sheets of paper. Currently the following ART marks are available for IG Unix:

- Arrow
- Attach-a-Note
- FREEHAND LINE
- HOLLOW ELLIPSE
- POLYLINE
- STRAIGHT LINE
- Typed Text

To save marks created for an image either:

- Save the annotations in a separate .ART file,
- "Burn in" the annotations permanently, or
- Save the annotations in the header of a TIFF image.

You may also combine these methods. You can, for example, leave classified information in an ART file and "burn in" non-classified information into the image. ART and ImageGear support the .ART file extension. You can print the resulting image with or without the marks.

Loading the ART Component

If you are going to use ART component functionality in your application you should attach ART component (“ART”) using ImageGear Component Manager API. Please, see the section Component Manager API for more detailes for this process as well as for the information how to make a call for ART component functions. Remember that you should make `#include` for ART public header `i_ART.h` in your application.

Working with ART Marks

Once you attach ART component you can annotate any image that you load into ImageGear. ART internally tracks the marks that are associated with an image. The end-user can save the image with the marks in a separate file with the .ART filename extension. Anytime the image is reloaded, a simple import call is used to “re-apply” the annotations.

You can also “burn in” the marks so they become part of the image and are not stored separately.

If the image is saved as a TIFF file, the annotations are automatically embedded in the TIFF file.

NOTE: This option can be disabled - See Saving Marks to File.

If a TIFF file with embedded annotations is loaded, the annotations are automatically “re-applied.”

ART annotations saved to a separate file (.ART) can be read only by software that supports .ART files, such as ImageGear .

Components of a Mark

All mark objects contain the following components:

- A Mark Attributes structure
- An index block
- A group block

The Mark Attributes structure is a set of attributes associated with a mark object. Attributes that do not apply are ignored.

The index block contains an unique number that is used for the mark object identification. When new mark object is created its index is generated (by incrementing 1 to the number just assigned) and stored in the global index block.

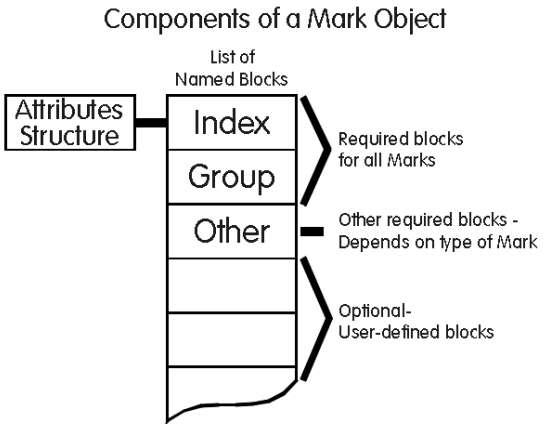
This unique number (32-bit unsigned integer) is used to refer to the mark in many of ART's operations. It cannot be modified.

The group block contains a `NULL`-terminated string that acts as the name of the block. It is initialized to the value contained in the global group block. If you have not set the global group block, a default value of `“(Untitled)”` is used. See the section ART Component Important Structures and Types or `i_ART.h` for more information about these structures.

ART refers to all data blocks associated with marks as “named blocks.” Each data block has a unique 8-byte name for identification. All names that are less than 8 bytes are padded with NULLs. The ART component uses two kinds of named blocks:

- mark blocks
- global blocks

Mark blocks contain data for individual marks. Global blocks contain data that is used to define the next mark created, as well as to store data that is not associated with an individual mark.



This diagram above shows the components of a mark. Central to any mark is the Mark Attributes structure. In addition, each mark contains a pointer to a list that includes the index and group blocks and other mandatory blocks, that vary depending on the type of mark, and an unlimited number of user-defined blocks. User-defined blocks contain data that might be useful to the developer.

You can define as many user-defined blocks as you want. Information that might be stored in a user-defined block includes:

- who last edited the file or mark
- user preferences, such as preferred line thickness

Note that the block shown in the list can occur in any order.

When a mark is created using **`ART_mark_create()`**, which is a lower-level method of creation, the group and index blocks are created automatically; any other mandatory blocks pertaining to the type of mark created need to be added one at a time using the **`ART_mark_block_create()`** function.

The following named blocks are defined by ART in the `i_ART.h` file. Each of these ones have a corresponding global block to hold default values for toolbar created marks:

ART_BLOCK_ANODAT	Used by many mark types to hold points and rotation information

ART_BLOCK_FILNAM	Name of file used for Image Reference, Text from File, and Audio
ART_BLOCK_DIB	Embedded DIB image
ART_BLOCK_GROUP	Name of group the mark belongs to
ART_BLOCK_INDEX	Unique index number for mark
ART_BLOCK_ANTEXT	For text mark types. Orientation and length of text, in bytes.

Creating a Mark Using Low-Level Calls

If you need to create a mark without the end user, call **ART_mark_create()**. You supply this function a structure of type ART_MARK_ATTRIBUTES, and it returns a unique index to the mark.

ART_mark_create() automatically creates the index and group blocks for your object. You have to add other required named blocks (their types depend on the mark object) one at a time using **ART_mark_block_create_ex()**. To see what blocks are required by each mark type look at the section Mark Objects: Descriptions, Types and Attributes.

Modifying Marks Programmatically

To modify marks programmatically, you set the attributes for block data, and call **()** or **ART_mark_block_create_ex()** to update that information.

Whether a mark can be modified depends on its access setting. A mark has write-access or is read-only.

Deleting Marks

To delete a mark, call **ART_mark_delete()** with the index number you supply. **ART_group_delete()** When a mark is deleted, the unique index identifiers of the remaining marks are not changed.

The callback function **LPFNIG_ART_DELETENOTIFYFUNC** is used to create a function that is called each time the user tries to delete a mark. If you return TRUE from a callback of this type, the mark is deleted.

Clipboard Operations

NOTE: There is no such option under generic Unix OS as clipboard. ImageGear emulates this option, so you may use it only within ImageGear application.

ART marks can be cut from and copied to the clipboard as well as pasted into an image individually or in groups.

The names of all functions that handle clipboard operations begin with ART_edit_. This group includes functions that undo changes made to marks. See the next section for a discussion about performing undos.

ART_edit_copy()	Copies an array of marks to the clipboard
ART_edit_cut()	Cuts an array of marks to the clipboard
ART_edit_paste()	Pastes any available marks from the clipboard into the current HIGEAR image
ART_edit_paste_count()	Queries the clipboard for the number marks available for pasting

Undoing Edits

As with Clipboard operations, undo begins with ART_edit_. The scope is determined by what is recorded. ART component supplies the following functions for “undoing edits”:

ART_edit_undo_enable()	Enable or disable undo operations.
ART_edit_undo()	Reverses any changes made to any mark
ART_edit_undo_clear()	Clears all pending undos from memory
ART_edit_undo_count()	Returns (in lpUndoCount) the number of times that ART_edit_undo()can be called (you can limit your undo loop)
ART_edit_undo_record()	Enables or disables the recording of every action

The example below shows ART_edit_undo_record() being set “on” before each mark in the image is deleted. Once the deletion loop has been completed, recording is shut “off” by calling **ART_edit_undo_record()** a second time with the last argument set to FALSE

```
/* Record all the deletes so that when an ART_edit_undo()  */
/* is called, the deleted marks will be restored at once,  */
/* instead of being restored individually. */

ART_edit_undo_record(hIGear, TRUE);

ART_mark_selected_first(hIGear, &hMarkIndex);
while (hMarkIndex != ART_INVALID_ID)
{
hMarkDelete = hMarkIndex;
ART_mark_selected_next(hIGear, &hMarkIndex);
ART_mark_delete(hIGear, hMarkDelete);
}

/* Turn off undo recording */
ART_edit_undo_record(hIGear, FALSE);
```

```
ART_GUI_mark_paint(hIGear,
IG_GRP_DEFAULT, ART_INVALID_ID, hWnd, TRUE);
```

A call to **ART_edit_undo()** restores all marks that were modified while recording was on. This means that all deleted marks are restored at once. If **ART_edit_undo_record()** was not used, then ART_edit_undo() would have to be called many times to restore all of the deleted marks.

Iterating Through Marks

ART component provides two functions that iterate through all of the marks associated with the current HIGEAR image. **ART_mark_first()** returns the index of the first mark. **ART_mark_next()** returns the index of the next mark. To iterate through all marks, call ART_mark_first(), then call ART_mark_next() within a loop, with a line such as:

```
ART_mark_first(hIGear, &hMarkIndex);
while (hMarkIndex != ART_INVALID_ID)
{
...
   ART_mark_next(hIGear, &hMarkIndex);
}
```

ART returns ART_INVALID_ID when there are no more marks to iterate through.

As you iterate through the marks, you can pass the returned index of each mark to any of the mark modification functions.

Iterating Through Blocks

ART supplies functions that let you iterate through both global blocks and mark blocks. To iterate through **global blocks**, use **ART_global_block_first()** and **ART_global_block_next()**. To iterate through the blocks that belong to an individual mark, use **ART_mark_block_first()** and **ART_mark_block_next()**.

All of these block iteration functions set the block name to NULL when there are no more blocks to iterate through.

Saving Marks to File

When the set of marks belonging to a HIGEAR image are saved, they can be saved separately into a file with filename extension ".ART", burned permanently into the image, or embedded into an image file. You can embed marks into TIFF files only.

- To save the marks to an .ART file, call the **ART_file_export()**, method, supplying the name of the .ART file to export. If an .ART file exists you can set a flag to indicate whether the new file replaces or overwrites the existing file. You can also append new mark objects to the existing ART file. If the existing file has multiple pages you can specify which page the new file overwrites, or where it should be inserted.
- To burn the marks into the image, call the **ART_image_burn_in()** function. Marks can be burned in by group, list, selection, etc., depending on what setting you choose. See Chapter "ART Component API Function Reference"

for more detailed information.

If you are saving a TIFF file, the annotations are automatically saved as an embedded file. Use **ART_control_option_set()** with the appropriate setting if you do not want the annotations to be embedded into the file:

```
nErrcount=ART_control_option_set(hIGear,
ART_CONTROL_SAVE_ENABLE, FALSE)"
```

Importing Marks to an Image

An .ART file can be imported into any loaded HIGEAR image as long as the ART component is attached. If the .ART file to import has multiple pages, you can specify the number of the page to load. To import an .ART file for association with the current HIGEAR file call **ART_file_import()**.

ART Component Callback Functions

ImageGear ART Component provides ART callback functions to create, delete, modify, select and interact with marks, as well as perform other operations within your ImageGear ART application. These functions work just in the same way as other ImageGear callback functions. Please see the section Working with ImageGear Callback Functions in this Chapter.

The following ImageGear ART callback functions are available for developing your imaging application:

- **LPFNIG_ART_CREATENOTIFYFUNC**
- **LPFNIG_ART_DELETENOTIFYFUNC**
- **LPFNIG_ART_MODIFYNOTIFYFUNC**
- **LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC**
- **LPFNIG_ART_PREMODIFYNOTIFYFUNC**
- **LPFNIG_ART_SELECTNOTIFYFUNC**

To use any of ART callback functions, you should previously register it in your application code using the function **ART_callback_register()**.

Please see the detailed description for every of these callback functions in ART Component Callback Functions Reference section of ART Component API Function Reference Chapter.

ART Component Important Structures and Types

Some arguments in ART component function calls are pointers to the internal ART component structures and data types such as ART_MARK_ATTRIBUTES structure that determines general attributes used by every ART mark. There are such important data types as ART component named blocks (ART_NAMED_BLOCK) that determine, for instance, mark's coordinates and orientation (ART_BLOCK_ANODAT) or the name of file for image marks (ART_BLOCK_FILNAM). For the full list of structures and data types currently provided by ART component please see i_ART.h file.

- You can see ART_MARK_ATTRIBUTES structure reference below:

Declaration:

```
typedef struct tagART_MARK_ATTRIBUTES {
DWORD dwType;
AT_RECT rcBounds;
AT_RGBQUAD rgbColor1;
AT_RGBQUAD rgbColor2;
DWORD bHighlighting;
DWORD bTransparent;
DWORD dwLineSize;
LONG lReserved1[2];
AT_LOGFONT lfFont;
LONG dwReserved2;
DWORD dwTime;
DWORD bVisible;
LONG lReserved3[11];
}
ART_MARK_ATTRIBUTES, FAR *LPART_MARK_ATTRIBUTES;
```

Structure Members:

dwType	The type of the mark: 3 = STRAIGHT LINE 4 = FREEHAND LINE 7 = Typed Text 10 = Attach-a-Note 502= POLYLINE 505= HOLLOW ELLIPSE 506= Arrow
rcBounds	Rectangle in image units. Can be a rectangle or two points.
rgbColor1	The main color; for example, the color of all lines, all rectangles, and standalone text.
rgbColor2	The secondary color; for example, the color of the text of an Attach-a-Note.
bHighlighting	If TRUE the mark is drawn highlighted. Highlighting performs the same function as a highlighting marker on a piece of paper. Valid only for lines, rectangles and freehand.
bTransparent	If TRUE the mark is drawn transparent. A transparent mark does not draw white pixels: transparent replaces white pixels with whatever is behind those pixels. Available only for images.
dwLineSize	The width of the line in pixels. **NOTE:** It is ignored when setting dwPenStyle for a straight line mark.
lReserved1[2]	Reserved; must be set to 0.
lfFont	The font information for the text, consisting of standard font attributes of font size, name, style, effects, and background color.
dwReserved2	Reserved; must be set to `0x0FF83F`.
dwTime	The time that the mark was first saved, in seconds, from 00:00:00 1-1-1970 GMT. Every annotation mark has time as one of its attributes. If you don't set the time before the file is saved, the time is set to date and time that the save was initiated. This time is in the form returned by the "time" C call, which is the number of seconds since midnight 00:00:00 on 1-1-1970 GMT. If necessary, refer to your C manual for a more detailed description.
bVisible	If TRUE the mark is currently set to be visible. Annotation marks can be visible or hidden.

Reserved3[11]	First 9 elements and Reserved[10] are reserved and must be set to 0. Value Reserved[9] is used by ImageGear to store DISABLE_PPM_FOR_FONTS mark attribute. Its interpretation is identical with global control parameter ART_CONTROL_DISABLE_PPM_FOR_FONTS but affects only one annotation. 2 helper macro-functions has been introduced to get and set this attribute. ■ AT_BOOL **ART_DISABLE_PPM_FOR_FONTS_GET**(LPART_MARK_ATTRIBUTES pMa); ■ AT_BOOL **ART_DISABLE_PPM_FOR_FONTS_SET**(LPART_MARK_ATTRIBUTES pMa, AT_BOOL bNewValue);

How to translate Windows fonts to Unix fonts (Typed Text, Attach-a-Note marks)

ImageGear ART font handling was originally designed to work on Windows platform. Annotation font is specified in the structure **ART_MARK_ATTRIBUTES** as **lfFont** field. This is **AT_LOGFONT** structure and represents ImageGear version of Windows **LOGFONT** structure described in Windows SDK. In order to work with Unix font some translation of Windows font specification is required.

It is performed in the following way:

Win32 font name	Font substituted by IG ART for Unix	Slant character (for lfItalic==TRUE)
"courier"	"adobe-courier"	'o'
"arial"	"adobe-helvetica"	'o'
"helvetica"	"adobe-helvetica"	'o'
"MS Sans Serif"	"adobe-helvetica"	'o'
"MS Serif"	"adobe-new century schoolbook"	'i'
"century"	"adobe-new century schoolbook"	'i'
"times"	"adobe-times"	'i'
"times new roman"	"adobe-times"	'i'

Slant field of Unix font name is set according to **lfItalic** field value and table above. If **lfItalic** is **FALSE** then slant character is **'r'**.

lfHeight field is translated from Windows logical units to respective value on Unix. It is assumed that **lfHeight** represents font size in display pixels with resolution of 96 DPI. **lfHeight** is converted to absolute units and then scaled according with required resolution (background image or X display) and current zoom value.

"Wildcarded" font name is created from family name, font weight and slant. All fonts matching specification are listed with **XListFonts** and then font with nearest **lfHeight** is used.

lfEscapement field is not handled in current version. For Windows ImageGear ART version valid values of this field are 0, 900, 1800 and 2700 that represent text rotated of 0, 90, 180 or 270 degrees. The same effect can be achieved with **ART_TEXTPRIVDATA** data block assigned with text annotation (**lOrientation** field).

Other **AT_LOGFONT** structure fields are not used in the current version.

Mark Objects: Descriptions, Types and Attributes

This section describes each mark available in ImageGear ART component, and its associated attributes.

NOTE: These constants define a range of user marks reserved for customer use. ImageGear will never use this range of the mark IDs for its purposes.

```
#define ART_MARK_USER_FIRST          1000
#define ART_MARK_USER_LAST           1999
```

Arrow

 The **Arrow** mark enables the end user to create an arrow over an image or document. It allows users to pointing to areas of a document that may be of interest to its reader.

Edit Mode: Users of the completed application can create this mark by clicking the **Arrow** button with the left mouse button, dragging the image, and then releasing. The head of the arrow is drawn at the point where the left mouse button is first pushed, and the tail (end) of the arrow is set at the location where the left mouse button is released. Pressing the right mouse button after the mark is created activates the popup edit menu.

Description	A line with a defined starting and ending point. An arrow head appears at the starting point.
Attributes	`rgbColor1`, `bHighlighting`, `dwLineSize`
OiGroup	String
OiIndex	String

OiAnoDat	ART_ARROW. This structure includes the following members: ■ lArrowType - Type of arrow head; valid values are 0 (ART_ARROW_POINTER), 1(ART_ARROW_SOLID), 2 (ART_ARROW_OPEN) and 3 (ART_ARROW_POINTER_SOLID) ■ lArrowAngle - The angle from the line to the outside edge of the arrow head. 15 is the default ■ lArrowLength - Length in image units of the arrow head. 25 is the default ■ lArrowEnd - Which end of the line the arrow head is to appear. Set to 0 for first point, set to 1 for second point ■ lReserved - Set to 0 ■ apPoints - End points of the arrow

Attach-a-Note

 The **Attach-A-Note** mark enables your end users to add notes to their annotated image or document.

Edit Mode: In the edit mode, end users create this mark by clicking the **Attach-A-Note** button with the left mouse button and then dragging the resulting rectangle over the area to be annotated.

Pressing the right mouse button activates the popup menu that enables users to change the font or the background colors of the note.

The **Choose Font** option displays a dialog box that allows users to change text properties. If the application is in Edit mode, and an end user wants to modify the text, they simply double-click the left mouse button when the cursor is over the note. The cursor appears at the beginning of the text to allow editing.

The **Set Background Color** option enables end users to change the color of the note. When this option is selected, a dialog box appears with a range of options.

Description	A colored rectangle that contains text.
Attributes	rgbColor1, rgbColor2, lfFont
OiGroup	String
OiIndex	String
OiAnText	ART_TEXTPRIVDATA. This structure includes the following members: ■ lOrientation - Angle of text baseline to image in tenths of a degree; valid values are 0, 900, 1800, 2700. ■ lAlign - See ART_TEXT_xxx bitflags ■ lReserved - Not in use ■ lAnoTextLength - 64K byte limit (32K for multi-byte data) for Attach-a-Note, typed text, text from file; 255 byte limit for text stamp ■ szAnoText - Text string for text mark types

Freehand Line

 The **Freehand Line** mark enables the end user to use freehand drawing for annotation. With this mark, end users create drawings as though they are using a pen and paper.

Edit Mode: In Edit mode, end users create this mark by clicking the **Freehand Line** button with the left mouse button, dragging the image, and then releasing the left mouse button. Pressing the right mouse button after the mark is created activates the popup edit menu.

Description	A series of lines where the starting point of line n+1 is the same as the ending point of line n.
Attributes	`rgbColor1`, `bHighlighting`, `dwLineSize`
OiGroup	String
OiIndex	String
OiAnoDat	`ART_POINTS`. This structure includes the following members: ■ `lMaxPoints` - The maximum number of points; must be equal to the value of nPoints. ■ `lPoints` - The current number of points ■ `ptPoint` - Points marking the beginning and ending of the line segment(s); in image (not scaled) coordinates relative to the upper left corner of `rcBounds` in `ART_MARK_ATTRIBUTES`.

Hollow Ellipse

 The **Hollow Ellipse** mark enables the end user to create a hollow ellipse to surround important areas.

Edit Mode: In Edit mode, end users create this mark by clicking the **Hollow Ellipse** button with the left mouse button, dragging the mark to the appropriate size, and then releasing the left mouse button. Pressing the right mouse button after the mark is created activates the popup edit menu.

Description	A hollow ellipse whose boundaries are defined by a rectangle. Two points define the rectangle, an upper left point and a bottom right point.
Attributes	`rgbColor1`, `bHighlighting`, `dwLineSize`
OiGroup	String
OiIndex	String

Polyline

 The **Polyline** mark enables the end user to create several connected straight lines by dragging and clicking the mouse.

Edit Mode: To create a polyline mark in an application, end users:

1. Click the left mouse button at the point where the polyline starts.
2. Drag the left mouse button to the location of a vertice.
3. Drag and click the left mouse button to create each vertice you want to add.
4. Move the cursor to the location where the polyline ends.
5. Double-click the left mouse button.
6. The polyline is complete.

Pressing the right mouse button after the mark is created activates the popup Edit menu.

Description	A series of lines such that the starting point of line *n*+1 is the same as the ending point of line *n*.
Attributes	`rgbColor1`, `bHighlighting`, `dwLineSize`
OiGroup	String
OiIndex	String
OiAnoDat	`ART_POINTS`. This structure includes the following members: ■ `lMaxPoints` - The maximum number of points; must be equal to the value of nPoints. ■ `lPoints` - The current number of points ■ `ptPoint` - Points marking the beginning and ending of the line segment(s); in image (not scaled) coordinates relative to the upper left corner of `rcBounds` in `ART_MARK_ATTRIBUTES`.

Straight Line

 The **Straight Line** mark enables the end user to create straight lines on a document or image.

Edit Mode: To create this mark, end users:

1. Click on the **Straight Line** button with the left mouse button.
2. Click the left mouse button at the point where the line shoud begin.
3. While holding the left mouse button down, drag the line to the point at which the line shoud end.

4. Release the left mouse button to complete the line.
5. Press the right mouse button after the mark is created to activate the popup Edit menu.

Description	A line with a defined starting and ending point.
Attributes	`rgbColor1`, `bHighlighting`, `dwLineSize`
OiGroup	String
OiIndex	String
OiAnoDat	`ART_POINTS.` This structure includes the following members: ■ `lMaxPoints` - The maximum number of points; must be equal to the value of nPoints. ■ `lPoints` - The current number of points ■ `ptPoint` - Points marking the beginning and ending of the line segment(s); in image (not scaled) coordinates relative to the upper left corner of `rcBounds` in `ART_MARK_ATTRIBUTES`.

Typed Text

 The **Typed Text** mark enables the end user to add descriptive text to their annotated image or document. The text can be created in various fonts, sizes, colors. The text appears over the original image or document.

NOTE: If you wish to rotate the mark, you must use a Windows True Type font (this is a Windows limitation).

Edit mode: End users create this mark by clicking the **Typed Text** button with the left mouse button and then dragging the resulting rectangle over the area to be annotated. This rectangle helps to position of the Text annotations. Text annotations are easily modified when the application is in the **Edit** mode. To modify it, the end user double-clicks the left mouse button when the cursor is over the text mark. The cursor appears at the beginning of the text allows editing.

NOTE: If end users create a text mark without any text, with a Transparent background, the resulting mark will exist but will not be visible.

Description	A series of characters entered from the keyboard.
Attributes	`rgbColor1`, `lfFont`

OiGroup	String
OiIndex	String
OiAnText	ART_TEXTPRIVDATA. This structure includes the following members: ■ lOrientation - Angle of text baseline to image in tenths of a degree; valid values are 0, 900, 1800, 2700. ■ lAlign - See ART_TEXT_xxx bitflags ■ lReserved - Not in use ■ lAnoTextLength - 64K byte limit (32K for multi-byte data) for Attach-a-Note, typed text, text from file; 255 byte limit for text stamp ■ szAnoText - Text string for text mark types

- **ICC_PROFILE_MODE**

ICC_PROFILE_MODE

Group: Control Parameter

Data type:

INT

Purpose:

Specifies the usage mode for ICC profiles embedded into JPEG2K and JPX images.

There are two possible values of this control:

- JPEG2K_ICC_PROFILE_LOAD: ImageGear loads the embedded profile and attaches it to the image. It does not apply the profile to image pixels during loading. If ImageGear ICC support is enabled, ImageGear uses this profile for image display and for colorspace conversion.
- JPEG2K_ICC_PROFILE_APPLY: ImageGear converts the image to sRGB colorspace using the embedded profile, during loading. Use this mode for compatibility with ImageGear version 15.1 and earlier.

Valid Values:

- JPEG2K_ICC_PROFILE_LOAD = 0
- JPEG2K_ICC_PROFILE_APPLY = 1

Default Value:

JPEG2K_ICC_PROFILE_LOAD

Using ImageGear MD Component

The MD (Medical) component is a full-featured ImageGear component. It supports the DICOM format, contains a custom API, and includes expanded image processing capabilities beyond those of the baseline ImageGear library.

NOTE: The ImageGear MD Component is supported by Linux platforms only.

Loading and Saving DICOM Images

With the MD component loaded, your application is ready to load and save DICOM image files in addition to the file formats normally supported by ImageGear.

Supported features:

- `IG_FLTR_DETECTSUPPORT` - autodetection.
- `IG_FLTR_PAGEREADSUPPORT` - single page file reading.
- `IG_FLTR_PAGEINSERTSUPPORT` - single-page file writing.
- `IG_FLTR_PAGEDELETESUPPORT` - page deleting from multi-page files
- `IG_FLTR_MPAGEWRITEPSUPPORT` - multi-page file writing.

Supported Compressions for Images:

- `IG_COMPRESSION_NONE` - 8, 9-16, 24 bpp
- `IG_COMPRESSION_JPEG` - 8, 9-16, 24 bpp
- `IG_COMPRESSION_JPEG2K` - 8, 9-16, 24 bpp
- `IG_COMPRESSION_RLE` - 8, 9-16, 24 bpp

Loading DICOM images

Load DICOM images in the same way as you load images of all other formats, using one of the Formats Component loading methods. You can use Page or Document loading modes, and all **Image Sources** that are supported by the Formats Component. Use DICOM Filter Control Parameters to control loading of the DICOM images.

Saving DICOM images

There are several ways to save DICOM images using ImageGear Medical. The functions included as part of the MD component API have a number of DICOM specific parameters. The others are included in the baseline API. You may need

to pass the DICOM parameters to these functions via the Image Control settings. You can also use the ImageGear's Multipage Image API to load and save multipage (cine) DICOM images. The saving functions are listed below.

Description	DLL
DICOM-specific saving function	**MED_DCM_save_DICOM**()
DICOM-specific saving function for a file that has already been opened and for which you have a File Descriptor handle	**MED_DCM_save_DICOM_FD**()
One of the baseline ImageGear saving functions (properties)	IG_save_file() IG_save_FD() IG_save_mem() IG_fltr_save_file()

The important consideration in writing a DICOM image is whether the image being saved out was originally a DICOM image. If the image was originally a DICOM image (and assuming that you have not altered its Data Set to contain illegal values), the saving of such an image is quite a simple process. This chapter describes how to save an image that has a valid Data Set, how to use the DICOM-specific saving functions, and what Image Control options can be used to save the image when using the baseline functions.

Using the DICOM specific saving function

MED_DCM_save_DICOM() saves the DICOM file using the name specified by the first argument. Here is a sample call:

```
MED_DCM_save_DICOM("OmyBack.dcm", hIGear,
   MED_DCM_TS_JPEG_LOSSY_8, TRUE, TRUE,
   MED_DCM_PLANAR_PIXEL_BY_PIXEL, TRUE, 100, 0);
```

Notice that the setting for Transfer Syntax in the sample call indicates that JPEG compression should be used on the file. For this reason the second-to-last argument **nJPEGQuality** is used. This is the quality setting for lossy JPEG compression. The setting shown (100) results in the highest quality that is possible using this compression scheme. If another Transfer Syntax had been specified, the setting of **nJPEGQuality** would have been meaningless.

Note that DICOM is a multipage format. When you save an image into existing DICOM file, ImageGear tries to append this image at the end of the file. If for some reason appending of the image is not possible, ImageGear returns an error. If you do not want to append a page, delete the existing file before saving to that filename, or (in DLL version) use **IG_fltr_save_file()** with **bOverwrite** parameter set to **TRUE**.

Saving a DICOM Image Using a File Descriptor

If you are saving a DICOM image to a file for which you have a File Descriptor handle, call **MED_DCM_save_DICOM_FD()**. Here is a sample call:

```
MED_DCM_save_DICOM(fd, hIGear, MED_DCM_TS_EXPLICIT_VR_LE,
FALSE, TRUE, MED_DCM_PLANAR_PIXEL_BY_PIXEL, TRUE, 0, 0);
```

Saving a DICOM Image Using Baseline ImageGear API Calls

ImageGear baseline API functions can also be used to save a DICOM image. However as these functions do not contain any of the custom parameters for saving a DICOM image, you may need to make some Filter Control calls first. You will find the list of available DICOM filter control parameters for saving in the section DICOM Filter Control Parameters.

Here is an example of setting DICOM filter control parameters for saving and then calling the baseline saving function. All of the parameters values used below are different from the component's default values:

```
/* get current value of SAVE_SYNTAX control parameter */
IG_fltr_ctrl_get(IG_FORMAT_DCM, "SAVE_SYNTAX", FALSE, NULL,
NULL, (LPVOID)nSaveSyntax, sizeof(nSaveSyntax));

/* set new value to SAVE_SYNTAX control parameter */
IG_fltr_ctrl_set(IG_FORMAT_DCM, "SAVE_SYNTAX",
(LPVOID)MED_DCM_TS_EXPLICIT_VR_LE, sizeof(nSaveSyntax));


/* get current value of SAVE_SMALLEST control parameter */
IG_fltr_ctrl_get(IG_FORMAT_DCM, "SAVE_SMALLEST", FALSE, NULL,
NULL, (LPVOID)bSaveSmallest, sizeof(bSaveSmallest));

/* set new value to SAVE_SMALLEST control parameter */
IG_fltr_ctrl_set(IG_FORMAT_DCM, "SAVE_SMALLEST", (LPVOID)TRUE,
sizeof(bSaveSmallest));
nErrcount = IG_save_file(hIGear, "knee.dcm", IG_SAVE_DCM);
```

Saving out a DICOM Image from a Non-DICOM Image

If you have created a Data Set for a loaded non-DICOM image and wish to save it, please remember that it is up to you to add all the Mandatory Data Elements to make it a valid Part 3 DICOM file. When you create the Data Set, a handful of basic Data Elements are added to it. These values were taken from the image. Note that they are not sufficient to satisfy the requirements of Part 3 of the Standard.

Saving a Multiframe DICOM Image

ImageGear MD component allows to save single-frame as well as multiframe (multipage) images. Saving a multiframe DICOM image is pretty straightforward. Use any standard image saving function from ImageGear baseline API. Image saving functions from Medical API do not support saving of multiframe images.

Use the following steps to create a multiframe image:

- Create or load first frame into **HIGEAR**.
- Create the DataSet if it is not present.
- Add the **DCM_TAG_NumberOfFrames** tag to the DataSet, if it is not present. Value of this tag is irrelevant but its presence is necessary.
- Save the image using any ImageGear baseline saving function.
- * Create or load second frame into **HIGEAR**.
- Create Data Set if it is not present.

- Add the **`DCM_TAG_NumberOfFrames`** tag to the Data Set, if it is not present.
- Save the image, specifying the same file name that you used for the first frame.

*** Repeat for all consequent frames.**

Several limitations that are imposed by the DICOM standard are listed below.

All frames of the image should have same dimensions, bit depth and photometric interpretation. All frames should use the same Transfer Syntax (compression). ImageGear returns an error if these conditions are not met.

NOTE: It is important to note that only a few DICOM modalities allow the presence of multiple frames in the image, while the others do not. When you try to add a frame to an existing DICOM file, ImageGear checks the file for presence of the NumberOfFrames (0028, 0008) Data Element. If it is present ImageGear tries to append the frame, otherwise ImageGear considers the file as belonging to a single-frame modality and returns an error.

Another important fact about DICOM format is that it has a continious structure which means all the Data Elements as well as image frames go one after another. This results in the following limitations:

- Inserting a frame in a file that contains multiple frames, would lead to rewriting all the consequent frames. The same thing would happen when overwriting a frame in a file that uses some compression. Changed size of the compressed frame would lead to rewriting the whole file. To prevent such unpredictable time-consuming operations, ImageGear Medical only supports appending a frame at the end of the DICOM image, but not insertion or overwriting of frames.
- ImageGear Medical writes the DICOM DataSet only when writing the first frame, and does not modify it when adding frames. The only exclusion is the "Number Of Frames" Data Element, which is always kept consistent with the actual number of frames.
- The "Number of Frames" Data Element has a Value Representation of "Integer String", i.e., it is stored in the file as a character string rather than an integer. This string can occupy up to 14 characters. Now imagine that you are appending frame to a file that has 99 frames. The number 99 occupies two bytes in the file (because it is composed of two digits). After a frame is added, the Number Of Frames DE becomes 100, needing 4 positions for its storage (three digits plus padding). This results in the need of shifting the whole file by two bytes. To prevent this, ImageGear Medical always leaves the maximum of 14 positions for storing the Number Of Frames DE. Moreover, it does not append a frame to a file where Value Length of the Number of Frames Data Element is less than 14 (for example, if the file has been created by some other vendor1).

If you need to insert or delete a frame, or modify DataSet in the DICOM image, create a new image with the modified DataSet, copy necessary frames to it, and then delete the original image.

Saving DICOM Images Using ImageGear's Multipage Image API (DLL only)

You can use ImageGear's Multipage Image API calls (IG_mpi_... and IG_mpf_...) for loading and saving multiframe DICOM images. Just note that ImageGear Medical only supports the appending of frames but not insertion, deletion or overwriting of frames.

DICOM Filter Control Parameters

The Medical Component provides the following filter control parameters for loading DICOM images:

- **LOAD_CONCAT_REPEATED_DE**
- **LOAD_CONVERTTO8G**
- **LOAD_DETECTSKIPDIMSE**
- **LOAD_MASKALPHACHANNEL**
- **LOAD_PAGENUMBER**
- **LOAD_SYNTAX**
- **LOAD_USE_8x8_LUT**
- **LOAD_USE_AUTO_WL_FOR_8G**

and the following filter control parameters for saving DICOM images:

- **SAVE_ASPART10**
- **SAVE_GROUPLENGTHS**
- **SAVE_JPEGQUALITY**
- **SAVE_LARGEST**
- **SAVE_PLANARCONFIG**
- **SAVE_SMALLEST**
- **SAVE_SYNTAX**

The Medical Component uses JPEG and JPEG 2000 format filters for loading and saving compressed DICOM images. Control parameters of JPEG and JPEG 2000 filters are taken into account when loading or saving such images (see JPEG section and JPEG2000 Component API Function Reference Chapter).

LOAD_CONCAT_REPEATED_DE

Group: Control Parameter

Description:

If this parameter is TRUE then concatenate duplicated tags data, FALSE otherwise.

Type:

Boolean

Default Value:

FALSE

Settings:

TRUE, False

LOAD_CONVERTTO8G

Group: Control Parameter

This parameters is not used any longer. Use medical image processing functions instead.

LOAD_DETECTSKIPDIMSE

Group: Control Parameter

Description:

This option controls loading process if a DICOM image contains DIMSE commands. DIMSE commands are a type of Data Element, with a group number of “0000” that are almost always removed by the DICOM network protocol before a transmitted image is saved to a disk file. However, sometimes they are found in the file and in such case ImageGear doesn’t automatically recognize the file as a DICOM image. This is done because the DIMSE Tags are very hard to differentiate from other file formats that ImageGear supports. However, if **DCM_CONTROL_LOAD_DETECT_SKIP_DIMSE** is set to **TRUE** then the auto format detection skips over the DIMSE Tags when it attempts to decide if the file is DICOM or not.

Type: Boolean

Default Value: False

Settings:

- False- ImageGear will not detect DICOM image files that contain DIMSE commands.
- True - Allow DIMSE commands and attempt to parse the file as a valid DICOM file.

LOAD_MASKALPHACHANNEL

Group: Control Parameter

Description:

This option controls what would have done if an Alpha Channel image has been stuffed into the upper unused bits of a 16-bit image (Bits Stored < 16). These extra bits can be masked off or loaded along with the actual pixel value. If they are not masked off, you may need to alter the 16x8 LUT in order to display the image appropriately.

Type: Boolean

Default Value: `True`

Settings:

- `True` - The extra bits (the Alpha Channel) are masked off.
- `False` - The extra bits will be loaded into the DIB with the rest of the pixel.

LOAD_PAGENUMBER

Group: Control Parameter

This parameter is not supported now. Instead, please use parameters of the Formats Component saving functions.

LOAD_SYNTAX

Group: Control Parameter

Description:

This parameter controls the types of DICOM files ImageGear attempts to detect. If the file that is being loaded does not fall into the category specified by this control parameter it will be ignored and a **`IGE_CANT_DETECT_FORMAT`** error will be returned.

Type: `enumIGMedTS`

Default Value: `MED_DCM_TS_AUTODETECT`

Settings:

- `MED_DCM_TS_AUTODETECT` - ImageGear makes its best to determine the format of the DICOM file.

- MED_DCM_TS_PART_10 - only files with Part 10 Header will be detected. The Transfer Syntax of the file will be determined from the header and used to load the remainder of the image file.

If you specify any standard DICOM Transfer Syntax, such as **MED_DCM_TS_IMPLICIT_VR_LE** or **MED_DCM_TS_JPEG_LOSSY**, the Medical Component will only load files having this Transfer Syntax.

LOAD_USE_8x8_LUT

Group: Control Parameter

Description:

If this parameter is TRUE then use 8x8 LUT for 8-bit images, otherwise use Palette.

Type:

Boolean

Default Value:

TRUE

Settings:

TRUE, False

LOAD_USE_AUTO_WL_FOR_8G

Group: Control Parameter

Description:

If this parameter is TRUE use auto window/level for 8g images, if VOI LUT is not present. If it is FALSE, use default 0...255 window/level.

Type:

Boolean

Default Value:

TRUE

Settings:

TRUE, False

SAVE_ASPART10

Group: Control Parameter

Description:

This parameter controls whether Meta Information Header is saved with the file, or not.

Type: Boolean

Default Value: `True`

Settings:

- `False` - The image is saved as a raw DICOM file
- `True` - The image is saved with a Part 10 header, making the file "Part 10-compliant".

SAVE_GROUPLENGTHS

Group: Control Parameter

Description:

This parameter controls the usage of Group Length values in a DICOM file. ImageGear treats these Data Elements as either on or off. That is they either are all included in each Group of Data Elements through the saving process or they are all absent. The internal Data Set that is attached to the **`HIGEAR`** does not contain Group Length Data Elements. When a DICOM file is to be written to disk they are computed and inserted if this parameter is set to **`TRUE`**.

Type: Boolean

Default Value: `True`

Settings:

- `True` - Group Lengths are saved
- `False` - Group Lengths are not saved

SAVE_JPEGQUALITY

Group: Control Parameter

This parameter is not supported now. Please use `QUALITY` control parameter of JPEG filter instead. Please see JPEG section in File Format Reference Chapter.

SAVE_LARGEST

Group: Control Parameter

Description:

Controls whether Largest Image Pixel Value (0028,0107) is updated by ImageGear.

If the original image Data Set did not contain a Data Element for Largest Image Pixel Value (0028,0107) and you set `SAVE_LARGEST = True`, ImageGear scans the image and determines a value for this DE. Largest Image Pixel Value is included in the Data Set of the DICOM image being saved and contains the ImageGear-determined value. The value of the DE from the original Data Set (if any) is ignored.

If you set `SAVE_LARGEST = False`, ImageGear does not determine this value for you, and the Data Set of the image being saved does not include the Largest Image Pixel Value Data Element. However, if the original Data Set did contain this DE, ImageGear preserves and includes it in the Data Set being saved

Type: Boolean

Default Value: `False`

Settings:

- `True` - The Largest Image Pixel value DE (0028,0107) are set to the minimum pixel value found in the image when it is saved.
- `False` - ImageGear does not update the Largest Image Pixel value DE when it saves a DICOM image.

SAVE_PLANARCONFIG

Description:

This parameter controls how the pixels are saved:

- in normal RGB order ("pixel by pixel" configuration)
- in a planar configuration meaning that all Red, Blue, and Green pixels are saved in separate planes

Type: AT_MODE

Default Value: `MED_DCM_PLANAR_PIXEL_BY_PIXEL`

Settings:

- `MED_DCM_PLANAR_PIXEL_BY_PIXEL` - Pixels are saved in normal RGB order
- `MED_DCM_PLANAR_PLANE_BY_PLANE` - Red, Blue, and Green are stored in

separate planes.

SAVE_SMALLEST

Group: Control Parameter

Description:

This parameter controls whether ImageGear updates the Smallest Image Pixel Value (0028,0106).

If the original image Data Set did not contain a Data Element for Smallest Image Pixel Value (0028,0106) and you set **`SAVE_SMALLEST = True`** ImageGear scans the image and determines a value for this DE. Smallest Image Pixel Value are included in the Data Set of the DICOM image being saved, and contains the ImageGear-determined value. The value of the DE from the original Data Set (if any) are ignored.

If you set **`SAVE_SMALLEST = False`** ImageGear does not determine this value for you, and the Data Set of the image being saved does not include the Smallest Image Pixel Value Data Element. However, if the original Data Set did contain this DE, ImageGear preserves and includes it in the Data Set being saved.

Type: Boolean

Default Value: `False`

Settings:

- `True` - The Smallest Image Pixel Value DE (0028,0106) is set to the minimum pixel value found in the image.
- `False` - ImageGear does not update the Smallest Image Pixel Value (0028,0106) when saving a DICOM image.

SAVE_SYNTAX

Description:

This parameter controls how a DICOM file is to be formatted when it is written to disk. It does not control the file being Part 10 or Raw (see **`bSaveAsPart10`**), but controls how the non-Group 2 Data Elements are formatted. It also specifies the compression that will be used for Pixel Data.

Type: `enumIGMedTS`

Default Value: `MED_DCM_TS_DEFAULT`

Settings:

- MED_DCM_TS_IMPLICIT_VR_LE - Implicit VR Little Endian: Default Transfer Syntax

- MED_DCM_TS_EXPLICIT_VR_LE - Explicit VR Little Endian
- MED_DCM_TS_EXPLICIT_VR_BE - Explicit VR Big Endian
- MED_DCM_TS_JPEG_LOSSY - JPEG baseline Transfer Syntax (process 01)
- MED_DCM_TS_JPEG_LOSSLESS - JPEG lossless non-hierachical Huffman transfer Syntax (process 14)
- MED_DCM_TS_JPEG_LOSSLESS_FIRSTORDER - JPEG Lossless, Hierarchical, First-Order Prediction
- MED_DCM_TS_JPEG_2K_LOSSLESS_ONLY - JPEG 2000 Lossless Only
- MED_DCM_TS_JPEG_2K - JPEG 2000
- MED_DCM_TS_RLE - RLE
- MED_DCM_TS_DEFAULT - Default transfer syntax: same as MED_DCM_TS_EXPLICIT_VR_LE
- MED_DCM_TS_UNKNOWN - Unknown Transfer Syntax
- MED_DCM_TS_PART_10 - This constant can be passed to the LOAD_SYNTAX image control parameter to load only those DICOM files that have a File Meta Information (Part 10) Header.
- MED_DCM_TS_AUTODETECT - This constant can be passed to the LOAD_SYNTAX image control parameter to load any detectible DICOM files.

Associating DICOM Data with an ImageGear Image

As a DICOM file is being loaded, the ImageGear Medical associates some additional structures with it. DICOM Data Elements are stored into two separate chunks: a table of Data Elements, which holds those Data Elements that make up the core of the image file, and a Part 10 Template, which holds the Data Elements from the Part 10 Header. The Part 10 Header is kept separate due to some of its special characteristics.

The table of Data Elements is internal and is only accessible via the API functions provided by this component. You can remove it (and free its memory), or create a new table if it does not exist. The names of the API functions that allow you to access the data in the internal Data Set table all start with **`MED_DCM_DS_`** .

We refer to the Part 10 Header area as a “template” because it should be created for every loaded DICOM image, even if the image's Data Set is empty. This template always contains the same set of storage fields for Data Elements—even if the Part 10 Header of the loaded image has no value for them. Further on in this section you will find information on how to get and set Part 10 Header values.

The Part 10 Template cannot be removed. However, as you are saving the image you can choose not to save it as “Part 10-compliant” and the image will be saved without this header.

Reading Data From Data Elements

The functions that return you the data for a DE's Data Field include:

MED_DCM_DS_curr_data_get()	Returns the data and size of the data
MED_DCM_DS_curr_data_get_string()	Returns the data as a string and returns the size of the string.

Both of these functions return you the contents of the Data Field of the Current Data Element.

To interpret the returned data call **`MED_DCM_DS_curr_info_get()`**. This function returns, among other things, the VR, the VL, and the Item Count.

If the data is binary, i.e. defined as an INT, WORD, etc., you can use the VL and Item Count to determine the length of each item in the Data Field. Binary items aren't delimited; they are stored end-to-end. Since all binary items must be of the same length, you can calculate the length of each item this way:

lengthOfEachItem = VL/Item_count

Using `Item_count` as a loop delimiter, you can now parse through the data by jumping `lengthOfEachItem` bytes for as many items as it's returned by `Item_count`:

```
for (item = 0; item < Item_count; item++)
   {
   int_array[item] = ((int *)value_field)[item];
   }
```

Character data items can be of either fixed length or variable length. In either case, you can forgo the calculation shown above for binary data because character data must be delimited by backslashes.

Writing Data to Data Elements

The following API functions allow you to write or overwrite the data of a Data field, respectively:

MED_DCM_DS_DE_insert()	Inserts a new DE to the Data Set. Its placement are determined automatically according to its Tag value. Your only power of placement is in specifying what level in the hierarchy of the Data Set to place the new DE.
MED_DCM_DS_curr_data_set()	Overwrites the Data Field of an existing Data Element.

Whether you are inserting a new DE or overwriting the Data Field of an existing DE, you may need to query the VR and VM by calling **`MED_DCM_util_tag_info_get()`**. With the VR in hand, you can call **`MED_DCM_util_VR_info_mode()`**which tells you the length that each item can have, and whether the items are of fixed or variable length. See the previous section "Reading Data From Data Elements" for how to interpret the value returned to you in VM.

The Components of the Data Set

A Data Set is composed entirely of Data Elements. In DICOM, all stored data, including the images themselves, is stored in fundamental DICOM building blocks called Data Elements.

Access to the Data Set is achieved by moving an internal Data Element index called the **"Current Data Element"** from Data Element to Data Element. There is a set of API functions, which just move the Current Data Element about the Data Set. Another group of API functions will allow you to retrieve or set the Value Field of the Current Data Element.

The Internal Data Set vs. the Original Data Set

The internal Data Set is similar to but not exactly like the Data Set found in the DICOM file. The DICOM standard has many options and various methods of storing its data. As the internal Data Set is being filled, the vital information is stored in an abstract form. The new form of this information makes it easier for you to work with. Most of this internal storage is completely transparent for you and your application, but there are a few items you should be aware of.

First of all, as you move to see the internal Data Set and inspect its contents you may notice that some items that were in the original Data Set are missing. One example of this are the "Group Lengths" (all of which have Element Numbers of 0000). Group Lengths which are optional in the DICOM specification, are placed in the Data Set to aid to quickly find certain Data Elements by allowing the parser to skip over large blocks of Data Elements that do not contain the Data Element being searched for. They serve no other purpose and are therefore removed. If you are going to save the Data Set back out to disk you may request that Group Lengths should be included in the new disk file. If you do so, ImageGear recalculates the Group Lengths in case any Data Elements have been added, removed or altered.

Also missing from the data is the Data Field for Pixel Data (7FE0,0010). It is the Data Element that holds the image. This Data Element is actually present but his Data Field is empty. This is because the image has been read and loaded into an ImageGear DIB.

Another difference from the internal Data Set and the original file version is that regardless of what the original encoding scheme was, the internal Data Set always are "Explicit VR." That is, the Value Representations (VRs) of each Tag are looked up in the Data Dictionary and recorded along with the other information. If you later write the Data Set to disk you may choose whether the Data Set should be Explicit or Implicit VR.

Critical Data Elements

"Critical Data Elements" are those DEs whose values are taken from the image not regarding to whether the Data Set has values for these DEs or not. Two examples of such DEs are the height (Rows) and width (Columns) of the image. Note that the Critical Data Elements in the internal Data Set are always kept consistent with the **`HIGEAR`**. For example, if you have resized the image, the **`DCM_TAG_Columns`** and **`DCM_TAG_Rows`** Data Elements are set to the new Width

and Height of the image. When an image processing function, such as Resize or Rotate, is applied to the DICOM image, the values for Rows and Columns will be updated from the **HIGEAR**.

The Hierarchy of the Data Set

A certain type of Data Element can be hierarchical meaning that it can have other groups of Data Elements "under" it similarly to the way files are stored under the directory structure. This special type of Data Element has a Value Representation of Sequence Delimiter (SQ). This Data Element l marks a set of Data Elements called "Items." This set can contain 0 or more embedded Data Elements. The hierarchical structure can make it difficult to keep track of where you are as you move through the Data Set. This is the reason of using the Current Data Element — to keep track of index you are positioned at in the array of Data Elements.

Data Set Levels

You may notice that many of the Data Set Navigation API contain a parameter of **level_op**. This argument tells whether the Data Set index should remain in the current level or can be moved to another level. The possible values for this argument are:

- MED_DCM_MOVE_LEVEL_FIXED
- MED_DCM_MOVE_LEVEL_FLOAT

Part 10 Header Access

When a DICOM image file is loaded it may or may not begin with a Part 10 Header. The Part 10 Header consists of 2 parts: a free-form block of 128 bytes called a Preamble, and a fixed list of 9 Data Elements, all of which have a Group Number of 0002. Both of these parts are stored in the Part 10 Template mentioned earlier. The use of the Preamble is up to the application and can be used to store anything you like, as long as it is 128 bytes or less. A blank or empty Preamble is indicated by 128 bytes of 0x00.

If the original image has a Part 10 header, the data from it is extracted and placed in the internal template mentioned earlier. If there is no header, the template is empty except for 2 fields: the File Meta Information Version (0002,0001) and the Transfer Syntax UID (0002,0010). Default values are assigned to these fields - File Meta Information Version is set to 0x0001, and Transfer Syntax UID is set to the Transfer Syntax detected by ImageGear when the image was loaded. The Version number is called for the DICOM specification and it value never need to be altered.

Access to the values in the Part 10 Template is done using a small set of methods or functions, which include either "Part10" or "preamble" in their names.

MED_DCM_DS_preamble_get()	get Part 10 Header Preamble
MED_DCM_DS_preamble_set()	set Part 10 Header Preamble
MED_DCM_DS_part10_get()	get Part 10 Header Data Elements, except Preamble

MED_DCM_DS_part10_set()	set Part 10 Header Data Elements, except Preamble

Displaying 16-bit Grayscale Images

Many of the medical images use more than 8 bits per grayscale pixel. The main concern in displaying such images is that commonly used monitors can display only 256 shades of gray, which corresponds to 8 bits per pixel. The 16-bit pixels need to be mapped in some way to 8-bit pixels. ImageGear uses two approaches for this mapping.

If a 9..16 bit grayscale image has been loaded from a non-DICOM file, its pixels are mapped to 8g by left shifting the pixels by (n-8), where n is the bit depth of the image. For example, if the image has 12 bits per pixel, its pixels will be left shifted by 4 bits, so the 8 most significant bits of the 12-bit pixel will be used. This mapping is done only for image display, and does not affect the image stored in memory. No 16-to-8 Look Up Tables (16x8 LUTs) are created for such images. These LUTs are discussed in the next paragraph.

When a DICOM image is loaded, a 16-bit to 8-bit internal display Look Up Table (16x8 LUT) is automatically created for it. This LUT gives more flexibility in displaying 9..16g images, allowing to display a certain range of pixel intensities with best contrast.

DICOM images may contain up to two Look Up Tables that describe how the image shall be displayed. “Modality” LUT specifies transform of image pixels into modality meaningful values, such as optical density or Hounsfield Units. “Value of Interest” LUT (VOI LUT) specifies what range of pixel intensities should be shown on the screen. Most often, both these LUTs are linear, and thus are presented by a pair of values that are similar to brightness and contrast. For Modality LUT, these are Rescale Intercept (0028, 1053) and Rescale Slope (0028, 1053). For VOI LUT, these are Window Center (0028, 1050) and Window Width (0028, 1051).

If all of these values are found in the file, they are all used to build the 16x8 LUT. If Rescale values are not found, default values (Intercept = 0.0, Slope=1.0) are substituted. If Window values are not found, the image is scanned for min and max pixel intensities, and the LUT is built to display the min intensity as black and max intensity as white. The values between min and max are linearly scaled between black and white.

You can adjust the values in the 16x8 LUT using a `MED_display_…()` function.

Pixel Padding Value

DICOM images sometimes contain a Data Element called “Pixel Padding Value” (PPV). The PPV is used mostly to fill in the corners of round images. DICOM provides a Tag for PPV which is (0028,0120). This Data Element stores a 16-bit gray-scale value that is to be treated as the Pixel Padding Value. Any pixels in the image that have this value are not to be treated as meaningful objects - but as background color.

When the ImageGear Medical loads a DICOM image that contains a PPV the value is captured and stored in the HDS, which is attached to the new image. In fact, 3 values are stored to the HDS: the PPV from the PPV Data Element, a flag indicating that a PPV was found in the file when it was loaded, and an 8-bit grayscale value used to display pixels with this value.

Use **MED_DCM_DS_PixPadVal_get()** and **MED_DCM_DS_PixPadVal_set()** functions to get/set the Pixel Padding Value that is to be used while displaying a 16-bit grayscale image.

Pseudocoloring of 16-bit images

Medical Display API contains methods for creating color LUTs that can be used to pseudocolor grayscale images.

DICOM images are displayed using functions from the baseline ImageGear API, and this may be all that you need if your image has a bit depth of 8 or 24. However, if your image depth is 16 bits per pixel, you may want to use one of the component's “display” API to fine-tune the 16x8 LUT for display. These functions have been written specifically for using with 16-bit images. Which of these functions you want to call and what parameters you want to use depends on what information you have about the image.

Here is a summary of the 16x8 LUT-altering API provided by the component:

MED_display_color_create()	Chooses a pre-defined ImageGear pseudo color scheme and creates 3 LUTs for the RGB components that are ready for use with display. You can call **MED_display_color_set()** to associate these RGB LUTs with an image.
MED_display_color_limits()	Displays over and under-saturated areas with pseudo color of your choice.*
MED_display_color_set()	Associates 3 LUTs with an image. Can be used to apply pseudo color to an image. You can set to your own LUTs, to the LUTs created by **MED_display_color_create()** , or set to **NULL** to use linear 0—255 gray-scale LUTs.
MED_display_contrast()	Displays using the Data Set settings for Rescale Slope and Intercept; Window Center and Width; and your own setting for Gamma correction.
MED_display_contrast_auto()	Same as **MED_display_contrast()** except that it will automatically derive Window Level and Center values for you, either by scanning the entire image or a rectangular sub-region of the image for the minimum and maximum pixel values.

Medical Component Grayscale Look Up Tables

The ImageGear Medical component provides a set of functions that allow you to create grayscale look up tables according to DICOM display attributes, such as VOI LUT, Modality LUT, Presentation LUT, etc.

Use the **MED_display_grayscale_LUT_build()** function to build a grayscale LUT from DICOM display settings. This function supports both linear and non-linear Modality and VOI LUTs, as well as presentation state related LUTs.

Once you have built the LUT, you can copy it to either the image, or to the image display settings, by using **IG_image_grayscale_LUT_update_from()** or **IG_dspl_grayscale_LUT_update_from()**.

IG_image_grayscale_LUT_... API internally uses the same LUT as the one that can be set by IG_display_option_get/set functions.

When ImageGear loads a grayscale DICOM image, it builds a grayscale LUT and attaches it to the image. This LUT can be obtained with the **IG_image_grayscale_LUT_copy_get()** function.

Example:

```
AT_MED_DCM_DISPLAY_SETTINGS DICOMDisplaySettings;
HIGLUT GrayLUT = (HIGLUT)NULL;

/* Initialize dicom display settings structure */
memset(&DICOMDisplaySettings, 0, sizeof(DICOMDisplaySettings));

if (MED_DCM_DS_LUT_exists(hIGear, DCM_TAG_ModalityLUT))
{
   MED_DCM_DS_LUT_copy_get(hIGear, DCM_TAG_ModalityLUT,
& DICOMDisplaySettings.ModalityLUT);
}
else
{
DICOMDisplaySettings.ModalityRescale.Slope = 1.0;
DICOMDisplaySettings.ModalityRescale.Intercept = 0.0;
}
DICOMDisplaySettings.VOIWindow.Center = 1024;
DICOMDisplaySettings.VOIWindow.Width = 2048;
DICOMDisplaySettings.Gamma = 1.0;

/* Build a grayscale LUT based on display settings */
MED_display_grayscale_LUT_build(&DICOMDisplaySettings,
&GrayLUT);

/* Copy LUT to the image. */
IG_image_grayscale_LUT_update_from(hIGear, GrayLUT);
```

See Also:

Working with Grayscale Look Up Tables

Grayscale Look Up Tables

Image Processing of 16-bit Images

Most of the baseline ImageGear processing functions support 9..16 bit images.

Medical Component presents a set of methods that were designed specifically for 9..16 bit and Medical images.

The Medical Image Processing API is described below

MED_IP_high_bit_transform()	Gives the high bit data element a new value and transforms the image accordingly.
MED_IP_histo_clear(),	Clears the histogram created by **MED_IP_histo_tabulate()**
MED_IP_histo_tabulate(),	Tabulates a histogram for a specified region of an image (or the whole image), and allows you to set the width of and number of bins that are used. This method also indicates whether the image is signed or not.
MED_IP_min_max(),	Gets the minimum and maximum pixel values from a specified region of an image (or the whole image), and also indicates whether the image is signed or not.
MED_IP_normalize(),	Converts a signed image to an unsigned image. Takes an argument for a minimum pixel value, converts the lowest pixel value in the image to the new minimum, and maps all the rest of the pixels in the image such that the original contrast is maintained.
MED_IP_promote_to_16_gray()	Promotes an 8g or 8i image to a 16-bit gray-scale image and gives you the possibility to select the position of the high bit.
MED_IP_reduce_depth_with_downshift(),	Reduces a 9—16-bit image to an 8-bit image using downshifting, that allows you to choose which 8 bits to use for the new image.
MED_IP_reduce_depth_with_LUT(),	Reduces a 9—16-bit image to an 8-bit image using your own LUT or the current display16x8 LUT.
MED_IP_swap_bytes(),	Corrects a poorly constructed image in which the bytes of each 16-bit pixel are in the wrong order.
MED_IP_contrast(),	Reduces a 9—16-bit image to an 8-bit image using Rescale Slope, Intercept, Window Center, Width and Gamma. You would normally use Rescale and Window settings from the Data Set, and your own setting for Gamma.
MED_IP_contrast_auto(),	Same as **MED_IP_contrast()** except that it automatically derives Window Level and Window Center values for you by scanning the image for the min/max pixel values.

Critical Data Elements

In this manual, we use the term “Critical Data Elements” for those DEs that are used to help load a DICOM image correctly. It's important to note that when certain IP methods are called, the toolkit alters the Data Set. For example, when an aspect of the image has been altered (for instance, by image processing) so, that the Critical DEs no longer correctly describe the image, the Data Set have to

be altered. When the image is actually saved, ImageGear also analyzes the DIB and set its own values for the Critical Data Elements of the file being saved. The original values are ignored.

The Critical Data Elements are:

(0028,0010)	Rows
(0028,0011)	Columns
(0028,0100)	Bits Allocated
(0028,0101)	Bits Stored
(0028,0102)	High Bit
(0028,0004)	Photometric Interpretation
(0028,0103)	Pixel Representation
(0028,0002)	Samples per pixel
(0028,0006)	Planar Configuration
(7FE0,0010)	Pixel Data

Working With DICOM Data Structures

When a DICOM image is loaded into ImageGear, it is loaded into a DIB just like any other supported image. In addition to the actual image, the following DICOM-specific data structures are also loaded: File Meta Information Header (if present) and the Data Set. This section explains how to read and/or manipulate the data in the Data Set or File Meta Information Header, or how to create your own Data Set or File Meta Information Header.

Getting and Setting Data Set and Part10 Header Data

With the exception of the first function listed below, these functions can be used for getting or setting various parts of the Part 10 Header data. These functions are useful when you are creating a new DICOM file.

The first function listed below simply returns some information about the Data Set attached to the **`HIGEAR`**. This returns the number of Data Elements in the Data Set.

The second function warrants a bit of explanation. When a DICOM image is loaded into ImageGear, the original Transfer Syntax is stored in the **`HIGEAR`**. This value is called the Original Transfer Syntax and cannot be changed. You can to read this information using **`MED_DCM_DS_orig_TS_get()`**. Even though you can alter the Transfer Syntax Data Element in the Part 10 Header template, the Transfer Syntax of the original image file is kept in case you ever need to know what it was. It does not affect the saving of a DICOM file. To set the Transfer Syntax for an image that you are saving use either the Image Control setting or fill in the TS parameter of **`MED_DCM_save_DICOM()`**.

`MED_DCM_DS_info_get()`	Returns the number of DEs in the Data Set

MED_DCM_DS_orig_TS_get()	Returns the Transfer Syntax used to create the image. It is returned as a **MED_DCM_TS_** constant.
MED_DCM_DS_part10_get()	Returns the data from a Part 10 item. You specify the item you would like to read. You must supply the item as **DCM_PART10_ITEM_** constant.
MED_DCM_DS_part10_set()	Sets the data of an item from the Part 10 meta-info header. You specify which item and supply it with new data to be stored to that item. You must supply a **DCM_PART10_ITEM_** constant.
MED_DCM_DS_preamble_get()	Gets the Preamble from the Part 10 header. Your receiving buffer must be at least 128 bytes. You could also use **MED_DCM_DS_part10_set()** with the appropriate constant to get this value.
MED_DCM_DS_preamble_set()	Sets the Preamble of the Part 10 header. You must supply it with the address of the Preamble and the length of the data you are saving to it. You could also use **MED_DCM_DS_part10_get()** with the appropriate constant to get this value.

Inserting a New Data Element

ImageGear Medical allows you to insert any number of Data Elements to an existing Data Set, or to a new Data Set that you have created by calling **MED_DCM_DS_create()**. (See the section "Creating a Data Set".)

DICOM Data Sets are sorted numerically by Tag number. Therefore, the Current Data Element does not affect the index position where your new Data Element is inserted. ImageGear takes your Tag number and automatically sorts the new DE into the correct position in the array. The Current Data Element does, however, determine the level at which the new DE is stored. For example, if the Current Data Element is in level 0 and has a Tag value of (0028, 0010) and your new DE has a Tag value of (0028, 0015), the new DE will be inserted into level 0 and positioned somewhere below the Current DE so that its unique Element Number fits numerically in ascending order.

NOTE: If a DE with specified Tag number already exists in the Data Set, it will be overwritten.

NOTE: If there are no DEs in the Data Set, the level for the new DE will be 0 or the "top level"

You have complete flexibility in what kind of DEs you can add to the Data Set. For each new DE, you have to supply ImageGear with a Tag structure, a VR, and the data. You can add it to any Group, to any position, with any VR, and with any data that you desire.

An example of using **MED_DCM_DS_DE_insert()**:

```
HIGEAR hIGear;
AT_DCM_TAG tag = 0x00111111;
AT_DCM_VR vr = MED_DCM_VR_CS;
char data[256] = “Data for test element“;
DWORD dwSizeOfData = strlen(data);

MED_DCM_DS_DE_insert(hIGear, tag, vr, data, dwSizeOfData);
```

This function requires you to supply the following values: a Tag, a VR, a Data Field, and the size of the data for the Data Field. Below are brief descriptions of what you would need to know in order to set these arguments.

Setting the Tag Value

All standard Tag values for a DICOM file are assigned constants in **`DataDict.h`**. You can also add your own user-defined Tag using **`MED_DCM_util_tag_info_add()`**.

Setting the VR

The DICOM specification assigns specific VRs for each defined public Tag. In other words, for a particular Tag, you must always use the specified VR. There are a few rare exceptions to this rule, and it is for this reason that you should tell **`MED_DCM_DS_DE_insert()`** what VR the data has.

If you don't know the VR for a particular Tag that you are adding, you can find out easily by calling the function **`MED_DCM_util_tag_info_get()`**, which returns you its VR as a constant.

Setting the Data for a VR of CS

There are no constants in ImageGear defined for those Data Elements which have a VR of CS. Refer to Part 3 of the DICOM specification for the valid Code Strings which you can enter for data of type CS.

Here is a sample call to insert a Data Element with a Tag of **`DCM_TAG_PhotometricInterpretation`**, which has a VR of CS (Code String). You must use one of the defined values from Part 3 of the DICOM specification. The allowed values are: **“MONOCHROME1”**, **“MONOCHROME2”**, **“PALETTE COLOR”**, **“RGB”**, **“HSV”**, **“ARGB”**, **“CMYK”**, **“YBR_FULL”**, **“YBR_FULL_422”**, and **“YBR_PARTIAL_422”**.

```
MED_DCM_DS_DE_insert(hIGear, DCM_TAG_PhotometricInterpretation,
MED_DCM_VR_CS,  "MONOCHROME2", 11);
```

Setting the Data Field

What kind of data can be stored to the Data Field depends on the Tag type, the VR, the restriction flags and the Value Multiplicity (VM) defined for the Tag. It is very helpful to call **`MED_DCM_util_VR_info_mode()`** or **`MED_DCM_util_VR_info_string()`** so that you can find out what kind of restrictions are placed on the kind of data that you would like to enter. Some types of Data Fields are tricky to work with. VRs of Person Name and Code String place a number of restrictions on what can go into the Data Field, and how it is to be formatted. For these VR types, you must consult Part 5 of the DICOM specification.

The **`lpData`** parameter is a binary buffer, which can contain a character string, a BYTE or WORD array, or one or several of numbers, such as Integers or Floats. Below there is an example how to set a value of VR=FD (double):

```
DOUBLE dblRefPixelValueX = 1.234;
MED_DCM_DS_DE_insert(hIGear,
DCM_TAG_ReferencePixelPhysicalValueX, MED_DCM_VR_FD,
&dblRefPixelValueX, sizeof(DOUBLE));
```

Setting the Length of a Data Field

Because some VRs allow a range of values, the length of all Data Fields cannot be assumed. For this reason, you must supply ImageGear with the length of your data.

Inserting a Sequence of Data Elements

Use the following steps to create and fill a sequence (SQ) of Data Elements:

1. Insert the Data Element of VR = SQ. This positions you one level lower in the DataSet hierarchy. At this level you can only add items to the sequence, not the Data Elements.

2. Insert the **`DCM_TAG_ItemItem`** Data Element to denote the beginning of a new item in the sequence. This positions you one level lower in the DataSet hierarchy. Now you can add Data Elements to the embedded Data Set.

3. Insert the data elements you would like to insert to this item.

4. Insert the **`DCM_TAG_ItemDeliminationItem`** Data Element to denote end of the item. This positions you one level higher. Repeat steps 2-4 for every item you would like to add.

5. Insert the **`DCM_TAG_SequenceDeliminationItem`** Data Element to denote the end of the sequence. This moves you one level higher - to the same level where you added the SQ Data Element.

Examples on creating a Data Element Sequence are below. Note that it is completely valid to add same Data Elements to different Items within a Sequence.

```
AT_DCM_TAG  tag;
char        data_str[64];

tag = DCM_TAG_ReferencedStudySequence;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_SQ, NULL, 0);

tag = DCM_TAG_ItemItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);

tag = DCM_TAG_ReferencedSOPClassUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.2");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str,
strlen(data_str));

tag = DCM_TAG_ReferencedSOPInstanceUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.2.1");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str,
strlen(data_str));

tag = DCM_TAG_ItemDelimitationItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
```

```
tag = DCM_TAG_ItemItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);

tag = DCM_TAG_ReferencedSOPClassUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.3");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str,
strlen(data_str));

tag = DCM_TAG_ReferencedSOPInstanceUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.3.1");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str,
strlen(data_str));

tag = DCM_TAG_ItemDelimitationItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);

tag = DCM_TAG_SequenceDelimitationItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
```

Deleting a Data Element

Data Elements can be deleted by calling **`MED_DCM_DS_curr_remove()`**. This function removes the DE designated by the Current Data Element. Therefore, you can use one of the `MED_DCM_DS_move_…()` functions to position the Current Data Element to the DE you want to remove.

TIP: If you've used **`MED_DCM_DS_move_find()`** or **`MED_DCM_DS_move_find_first()`** to position the Current Data Element, be sure to check the value of the Boolean argument to see whether you successfully found the Tag or Group Number that you were looking for. Otherwise, you might unintentionally delete the wrong DE.

Some calls to **`MED_DCM_DS_curr_remove()`**result in deletion of more than one Data Element:

- If you are going to delete a DE with a VR of SQ, all the items and consequently DEs below it will be deleted also.
- If you are going to delete a DE of type Item (FFFE, EOOO), all DEs below it will also be deleted.

The following types of DEs may not be deleted:

- Sequence Delimitation Item (SQD) – (FFFE, EODD)
- Item Delimitation Item (ID) – (FFFE, EOOD)

Trying to delete DEs of these types will have no affect on the Data Set. These type of DEs will be automatically deleted by ImageGear when their accompanying set of DEs are deleted.

Getting and Setting Data of the Current Data Element

Notice that all functions in this group include "curr" as part of their names. Below is a description of these functions..

MED_DCM_DS_curr_data_get()	Returns the data and size of data from the CDE. The data is returned in its native format. Use **MED_DCM_DS_curr_info_get()** to get the VR in order to know how to handle the data.
MED_DCM_DS_curr_data_get_string()	Same as **MED_DCM_DS_curr_data_get()** except that the data is always returned as a string.
MED_DCM_DS_curr_data_set()	Sets/overwrites the Value Field (data) of the CDE. Check the VR required. You cannot change the VR of an existing DE.
MED_DCM_DS_curr_index_get()	Returns the index of the CDE. Lets you know where you are in the Data Set.
MED_DCM_DS_curr_info_get()	Returns the following pieces of information about the CDE: Tag, VR, VL, Item Count. This is a very important method. Use it as a precautionary call before modifying info in the CDE or inserting/deleting.

Creating a Data Set

If you have a **HIGEAR** image to which you would like to attach a Data Set, call **MED_DCM_DS_create()**. This file might have been loaded from a non-DICOM format or may be a DICOM file in which you've destroyed the Data Set (see the section above).

This function creates an empty Data Set and attaches it to the ImageGear's image. The function creates a Data Set structure (outlined above) and fills it with a few Data Elements whose values can be derived from the image. This function also requires you to supply a Transfer Syntax that is saved in the Part 10 Header template as the original Transfer Syntax (as if it was loaded from a DICOM file). If you then save the image and specify a different Transfer Syntax, you overwrite the TS value specified at the time of Data Set creation.

Deleting a Data Set

If you have loaded a DICOM image file and do not need to keep the Data Set around you can free up the memory used to store all the Data Elements by calling **MED_DCM_DS_destroy()**. Once this function is called the remaining image is no different than any other ImageGear image that was created from any other non-DICOM format. The removed Data Set is discarded and cannot be recovered.

Creating a Part 10 Header

When a DICOM image is loaded with the ImageGear Medical, a Part 10 Header Template is initialized automatically. If you save the image as a Part 10-compliant image (which is the default option), the values from this template are used to fill the header saved with the file.

For those wishing to modify or read the values stored in the Part 10 Header Template, the component supplies 11 constants. All except the last one correlate directly to the fields found in a Part 10 Header. The last constant can be used by ImageGear to store the length of the Private Info field, and is for informational purposes.

Below is the list of Part 10 constants defined in DCM.h. Each one is followed by its size in bytes. A check appears next to each field that is considered "Mandatory" by the specification. ("Mandatory" is one of three possible "Types" for a DICOM DE. The other Types are "Optional", and "Mandatory Depending on some Condition".)

Mandatory	MD Component Constant	Size of Data Field
Yes	DCM_PART10_ITEM_PREAMBLE	128 bytes
Yes	DCM_PART10_ITEM_VERSION	2 bytes
Yes	DCM_PART10_ITEM_MSSOPCLASSUID	Max 64 bytes
Yes	DCM_PART10_ITEM_MSSOPINSTUID	Max 64 bytes
Yes	DCM_PART10_ITEM_TRANSSYNTAXUID	Max 64 bytes
Yes	DCM_PART10_ITEM_IMPLCLASSUID	Max 64 bytes
	DCM_PART10_ITEM_IMPLVERNAME	Max 16 bytes
	DCM_PART10_ITEM_SRCAETILE	Max 16 bytes
	DCM_PART10_ITEM_PRIVINFOCRUID	Max 64 bytes
	DCM_PART10_ITEM_PRIVINFO	any length
	DCM_PART10_ITEM_PRIVINFO_SIZE	DWORD - length of Priv Info - read only

Please note the following about the above constants:

- You can set **DCM_PART10_ITEM_PREAMBLE** to anything you want as long as it doesn't exceed 128 bytes
- **DCM_PART10_ITEM_VERSION** is set by the toolkit to a default value of 1. You should not change this.
- **DCM_PART10_ITEM_TRANSSYNTAXUID** – we set the value of this DE when the image is loaded. You can read this value; you can even set this value. But when the image is saved your setting are ignored.

If you enter data in **DCM_PART10_ITEM_PRIVINFO**, it is stored with the new Part 10 Header. Also, ImageGear calculates its size and saves this information internally. This read-only size information is not made part of the template but is kept in case you need to know how much memory to allocate when getting the information from **DCM_PART10_ITEM_PRIVINFO**. **DCM_PART10_ITEM_PRIVINFO_SIZE** isn't stored to the new Part 10 header.

Here is an example of a Part 10 field being populated:

```
char szPreamble[] = "This DICOM File is MINE!";
DWORD size_of_data;
size_of_data = sizeof(&szPreamble[0]);

MED_DCM_DS_part10_set(hIGear, DCM_PART10_ITEM_PREAMBLE,
&szPreamble[0], &size_of_data);
```

Updating DICOM DataSet without updating Pixel Data

There are cases when an application needs to update DataSet of a DICOM file, while leaving PixelData unchanged. For example, if the image is compressed using a Lossy compression scheme, this will avoid degradation of image quality.

Use **MED_DCM_DS_update_file**() function to update DataSet in a DICOM file, without changing its pixel data.

Working with DICOM Data Dictionary

The MD component encapsulates the DICOM Data Dictionary (Part 6) in a static and internal table, which the component uses to look up Tags as they are being decoded. This table is generated from the component header file **DataDict.h**. It can be used also by your application to provide the DICOM given name or other information about a given Tag. To retrieve this information use the **MED_DCM_util_tag_info_get()** function.

The Data Dictionary is also used to check the VR of Data Elements as they are being decoded. If a file is Explicit VR then as each Data Element is read, the VR found in the file is matched against the VR in the Data Dictionary. If it is determined that the VR is not appropriate for the Tag type, an error condition may occur.

You can add your own user-defined Tags to the Data Dictionary, or add new Tags that have been added to the DICOM specification so that you keep your application current. The Tags that you add, whether they are user-defined, or new to the DICOM specification, will be stored in a separate table in memory, rather than added to **DataDict.h** file. **DataDict.h** is reserved for public DICOM Tags. However, when your application is running, ImageGear Medical will treat the two tables as one, and will be able to process any valid Tag number that you reference. To add a Tag use **MED_DCM_util_tag_info_get()**.

Since the internal Data Dictionary tables are statically defined, any Tags you wish to add should be added to your initialization code since the table need to be updated each time the toolkit is started up.

The internal Data Dictionary holds several pieces of information for each Tag: it holds the VR, the VM, the version in which the Tag is last used, and the character string name of the Tag as it appears in the DICOM Standard.

The version information is stored because there are many Tags in the DICOM 3 Standard Data Dictionary that are now obsolete or “Retired” as DICOM calls them. If a Tag has been Retired in DICOM 3, then the version is stored as 2. If the Tag has not been retired then it is stored as 3 (for DICOM version 3). Retired Tags also include the string “(RET)” at the end of their name string.

Working with Overlays

Use **MED_overlay_apply**() function to load graphic overlays from DICOM DataSet to ART annotations. Use **MED_overlay_extract**() function to save ART annotations into DICOM DataSet as overlay data.

Working with Presentation State Objects

Presentation State (PS) objects serve for the following purposes: consistent display of images on various devices and media, storage of specific settings for display (contrast transformations, geometric transformations), and storage of annotations. They also allow for special display of multiframe images.

Presentation State object files do not include actual images, but reference one or more images.

Presentation State object files, in addition to LUTs, display parameters and annotations, contain other tags, such as Patient/Study/Series info, Referenced Image UIDs, etc. ImageGear provides read/write access to these tags.

Consistent display of images

DICOM standard introduces the Standardized Display System. A Standardized Display System may be a printer, a monitor or some other display device which has been calibrated according to the Grayscale Standard Display Function (GSDF). The main feature of such display system is that throughout its display range, equal differences in digital input values correspond to visually equal differences in luminosity. Two Standardized Display Systems will always show the same detail in an image, even if their physical characteristics are different.

The values that can be used as input to a Standardized Display System are called “Presentation Values” (“P-Values”). To map image pixel intensities into P-Values, Presentation Look Up Table (“P-LUT”) is used. It is applied after Modality and VOI LUTs.

Presentation LUT is stored in a Presentation State DataSet.

If a display device is not physically calibrated to comply with GSDF, but its characteristic curve (a table that lists luminosities for each digital input value) is known, it can be calibrated at the software level. In that case, P-Values shall be used as input to a GSDF LUT, which will map them to the device’s input values according to GSDF. Thus, a non-standardized device together with its GSDF LUT can be considered as Standardized Display System.

ImageGear Medical component allows to build a GSDF LUT from a device’s Characteristic Curve.

ImageGear uses all available LUTs (Modality LUT, VOI LUT, Presentation LUT and GSDF LUT), to build the general 16x8 or 8x8 LUT that maps image pixel values into display input values.

Grayscale contrast transformations

Presentation State DataSet may include VOI and Modality LUTs. Their usage is the same as in normal DICOM images. If either of these LUTs is present, it overrides the LUT found in the image.

Geometric transformations

Presentation State DataSet may include Rectangle of Interest, scale mode (True size, Scale to Fit, Magnify), and orientation (Rotate, Flip).

Annotations

Annotation objects include: Text box, Point, Polyline, Spline, Circle, Ellipse. Graphical objects can be filled or not filled. Annotations can be grouped into layers.

ImageGear Medical component uses ART marks to represent DICOM annotation objects. When a Presentation State DataSet is applied to an image, ImageGear creates ART marks for each of the annotation objects. These marks can then be edited with the ART API methods.

DICOM Graphic Annotation objects are grouped into Layers. All objects on the same layer have the same color (recommended grayscale value). ImageGear uses ART mark groups to represent layers.

When extracting presentation state from a HIGEAR that has ART marks attached to it, ART marks are converted to DICOM Graphic Annotation objects or Text Annotation objects. The table below lists ART marks that can be converted, corresponding DICOM Graphic Annotation objects, and properties that Graphic Annotation objects support. Marks that are not listed in this table are ignored.

ART mark type	DICOM Graphic Annotation Object	Supported properties
POINT	POINT	Point coordinates
POLYLINE	POLYLINE	Node coordinates, "Filled" property
FILLED_POLYGON	POLYLINE	Node coordinates, "Filled" property
CURVE	INTERPOLATED	Node coordinates, "Filled" property
FILLED_ELLIPSE	ELLIPSE	Bounds coordinates, "Filled" property
HOLLOW_ELLIPSE	ELLIPSE	Bounds coordinates, "Filled" property

The following ART marks are saved as DICOM Text Annotation objects:

ImGearARTMarkType	Supported properties
TYPED_TEXT	Text, bounds coordinates,
CALLOUT	Text, bounds coordinates, anchor point coordinates, anchor point visibility

Graphic Annotation objects do not have a color property. Instead, Graphic Layer has "Recommended Display Grayscale Value" property. ImageGear converts the color of the first mark in the ART group to grayscale and uses this value to initialize the "Recommended Display Grayscale Value" property of the layer.

Working with Presentation State Objects

You can load Presentation State files in the same way as you load normal DICOM images. However, since PS files do not contain an image, HIGEAR will be set to an empty image (DIB.biCompression |= IG_BI_EMPTY). Data Set of such image is accessible through Medical API. Such images can also be saved in the same way as normal DICOM images.

A HIGEAR containing a Presentation State DataSet can be “applied” to another HIGEAR that contains an image. This operation applies display and annotation settings from Presentation state onto the target image. In the opposite way, display settings and annotations can be exported from an image into another HIGEAR.

When you apply Presentation State to an image, ImageGear updates its 16x8 or 8x8 LUT using the Presentation LUT. However, it does not store Presentation LUT with that image. If you would like to work with Presentation LUT, you should allocate memory for it in your application. Pass Presentation LUT as a parameter to medical display functions. You can also save Presentation LUT to a Presentation State data set.

If a Characteristic Curve is available for a display, you can build a GSDF LUT from it, and use it in calls to medical Presentation State and Display functions. Otherwise, pass a NULL to GSDF LUT parameter.

Waveform Image Support

ImageGear supports both DICOM raster and vector (waveform) images. ImageGear loads waveform images in vector form. You can use Vector component API to modify waveform images that have been loaded to ImageGear. Use any image loading functions to load DICOM waveform images.

ImageGear reads and displays waveform data (graphs), as well as axes and some annotations.

As of this writing, ImageGear creates the following annotations for waveform images:

- X axis with second marks, according to SamplingFrequency tag value.
- X axis for every image, if it is visible and does not intersect with other images
- Y axis for all images.
- Minimum and maximum Y values for each graph

Using ImageGear PDF Component

This section tells how to use ImageGear PDF component.

- About the PDF Component
- How to Attach the PDF Component to Core ImageGear

- Working with PDF Layers
- Distributing PDF and PS Fonts with Your Application
- PDF Format Filter
- PDF Format Filter Control Parameters
- PostScript Filter Control Parameters

About the PDF Component

PostScript Language (PS)

The **PostScript Language (PS)** is a simple interpretive programming language with powerful graphics capabilities. Its primary application is to describe the appearance of text, graphical shapes, and sampled images on printed or displayed pages, according to the Adobe® imaging model. A program in this language can communicate a description of a document from a composition system to a printing system or control the appearance of text and graphics on a display. The description is high-level and device-independent.

Adobe® Portable Document Format (PDF)

The **Adobe® Portable Document Format (PDF)** is the native file format of the Adobe® Acrobat® family of products. PDF relies on the same imaging model as the PostScript page description language to describe text and graphics in a device-independent and resolution-independent manner. A document can be converted between PDF and the PostScript language; the two representations produce the same output when printed. To improve performance for interactive viewing, PDF defines a more structured format than that used by most PostScript language programs. PDF also includes objects, such as annotations and hypertext links that are not part of the page itself but are useful for interactive viewing and document interchange. However, PDF lacks the general-purpose programming language framework of the PostScript language.

Using the ImageGear PDF API allows you to load, save, edit and process native PDF and PostScript documents. The ImageGear PDF Component can also perform rasterization of PDF and PostScript documents, converting them to bitmaps. The component also provides you with the ability to extract text from loaded PDF and PostScript documents

ImageGear PDF component provides full multi-page reading and writing support for the entire document as well as specified set of pages. You can detect, read, write, append, insert, replace, swap and delete a specified page in the PDF document.

NOTE: The Adobe®-based PDF Component is applicable for Linux (i386), Solaris 32-bit, and AIX Unix Platforms.

Linux for PowerPC, HP-UX 32-bit & 64-bit, HP-UX for Itanium, and Sun Solaris 64-bit use the GS PDF Component .

Content Editing

Content editing provides an API for creating, accessing and editing PDF page content objects. With this API you can work with a page's content as with a list of such objects like images, texts, forms. Retain, modify and save their data and properties.

Document Fonts

Document font support includes:

- Listing the fonts available in the host system and finding a system font that matches a PDF font
- Creating a font from the system font and encoding as well as from the specified attributes
- Changing font's information
- Editing and embedding a font in the document
- Both single and multiple bytes fonts are supported

Word Extraction

Text words extraction includes:

- Extracting words from a PDF document or specified page
- Enumerating and sorting the words
- Getting word layouts, styles and characters

PDF Multi-thread Safety

Note that the access to the same PDF document from different multiple threads is not permitted because the multiple threads cannot share Adobe PDF Library datatypes. PDF document created/opened in the main thread can be only used from the main thread.

Document Metadata

You can get and set PDF document metadata corresponding to a document's Info dictionary.

Native Printing

Native PDF document printing now renders the document content directly to the printer, so it is fast and requires less memory.

NOTE: PDF support is compatible with Adobe® PDF version 1.5 as defined in the Portable Document Format Reference Manual Version 1.5, distributed by Adobe Systems Incorporated. It provides reading capability up to the PDF version 1.5 and writing of the PDF 1.5 documents.

NOTE: PostScript support is compatible with Adobe® PostScript® 3.0 language as defined in the PostScript Format Reference Manual, distributed by Adobe Systems Incorporated. It provides reading capability up to the PostScript 3.0 Language Level 3 and writing of the PostScript 3.0 files with the Language Level 1, 2 and 3. This includes writing of the Enhanced PostScript (EPS) files with standard and extended preview as well as w/o preview at all.

How to Attach the PDF Component to Core ImageGear

To use the ImageGear PDF Component, you have attach this component to core ImageGear using the following function:

```
IG_comm_comp_attach("PDF")
```

To check if the Component is attached successfully, use the following function:

```
IG_comm_comp_check("PDF")
```

This function (method) returns **TRUE** if the Component is attached.

Initializing the Component

To initialize the ImageGear PDF component, the "PDF" component needs to be attached to the ImageGear component manager.

The following resource content is required by the ImageGear PDF component initialization routine.

Resource\PDF\CIDFont\	PDF CID fonts directory
Resource\PDF\CMap\	PDF font CMaps directory
Resource\PDF\Font\	PDF fonts directory
Resource\PS\ColorRendering\	Color rendering PostScript
Resource\PS\ICCProfiles\	Directory containing the ICC profiles that allow using the Adobe Color Engine (ACE)
Resource\PS\Fonts\	PS fonts directory
Resource\PS\ProcSet\	PostScript procedures

Resource\PS\ps.vm	A file for initialization the PostScript Interpreter's virtual memory
Resource\PS\startupNORM.ps	Startup PostScript program used to initialize the PostScript Interpreter
Resource\PS\superatm.db	Adobe® Type Manager® (ATM®) database used to substitute missing fonts

The ImageGear PDF component uses the following PDF global control parameters to locate the resource content:

PDF.PDF_RESOURCE_PATH	path to the Resource\PDF directory
PDF.PS_RESOURCE_PATH	path to the Resource\PS directory
PDF.HOST_FONT_PATH	path to the system font directory
PDF.TMP_PATH	path to the TEMP directory

If the PDF global parameters are not defined, the ImageGear PDF does the following:

- Resource content gets from the directory defined by `IMAGE_GEAR_PDF_RESOURCE_PATH` and `IMAGE_GEAR_PS_RESOURCE_PATH` environment variables, "/usr/local/ImageGear15/Resource/PDF/" and "/usr/local/ImageGear15/Resource/PS/" if those are not defined
- Host fonts get from the directory defined by `IMAGE_GEAR_HOST_FONT_PATH`, "/usr/local/hfonts/" if that is not defined
- Temporary directory gets from the directory defined by `IMAGE_GEAR_TEMP_PATH`, "/tmp/"if that is not defined

PDF Global Control Parameters

PDF.ENABLE_POSTSCRIPT

Used to enable/disable PostScript support in PDF component.

By default the ImageGear PDF Component enables PDF and PostScript file format support. But it is possbile to disable PostScript file format support in case if it is not required. There is a boolean global parameter named `PDF.ENABLE_POSTSCRIPT` used to control this.

To disable PostScript use:

```
AT_BOOL     bEnable = FALSE;
IG_gctrl_item_set( "PDF.ENABLE_POSTSCRIPT", AM_TID_AT_BOOL,
&bEnable,
sizeof(AT_BOOL), NULL );
```

Retrieving Info About the Component

To retrieve information about the attached Component, call the following function:

```
IG_comm_comp_list(LPUINT *lpnCount, UINT nIndex, LPCHAR lpComp,
DWORD dwCompSize, LPUINT lpnRevMajor, LPUINT lpnRevMinor, LPUINT
lpnRevUpdate,
LPCHAR lpBuildDate, UINT nBDSize, LPCHAR lpInfoStr, UINT
nISSize)
```

This function (methods) provides you with the full list of info about the component determined by **`nIndex`** index from the list of currently attached components whose number are returned trough **`lpnCount`** argument.

For more detailed information about these functions usage see Using ImageGear Component Manager section

Working with PDF Layers

This section explains how to work with PDF Layers, as follows:

- Layer Objects and Visibility
- Layers Diagram
- Visibility Policy
- Working with Containers, Dictionaries and Layers using ImageGear PDF sample

For more information, see Using ImageGear PDF Component and **`HIG_PDF_DICTIONARY`** and **`HIG_PDF_LAYER`**.

Layer Objects and Visibility

The following objects are related to PDF Layers and responsible for the visibility.

Layers

Each layer has a name and a visibility state for the containers connected to the layer through a Dictionary. The visibility state can have one of two values - ON or OFF.

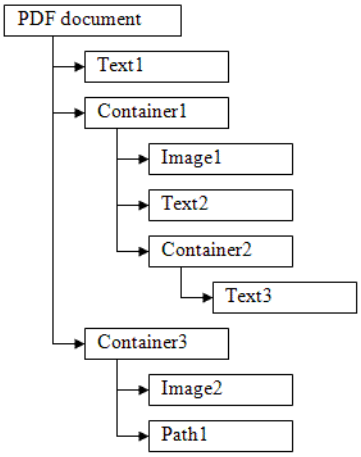
Dictionaries

Each dictionary contains the array of layers and a Boolean function, which takes all the layers' states as input and applies the visibility policy function to the layers' state values producing the Boolean result whether or not to display all the objects from the dictionary's container. The output can have one of two values - ON or OFF.

Containers

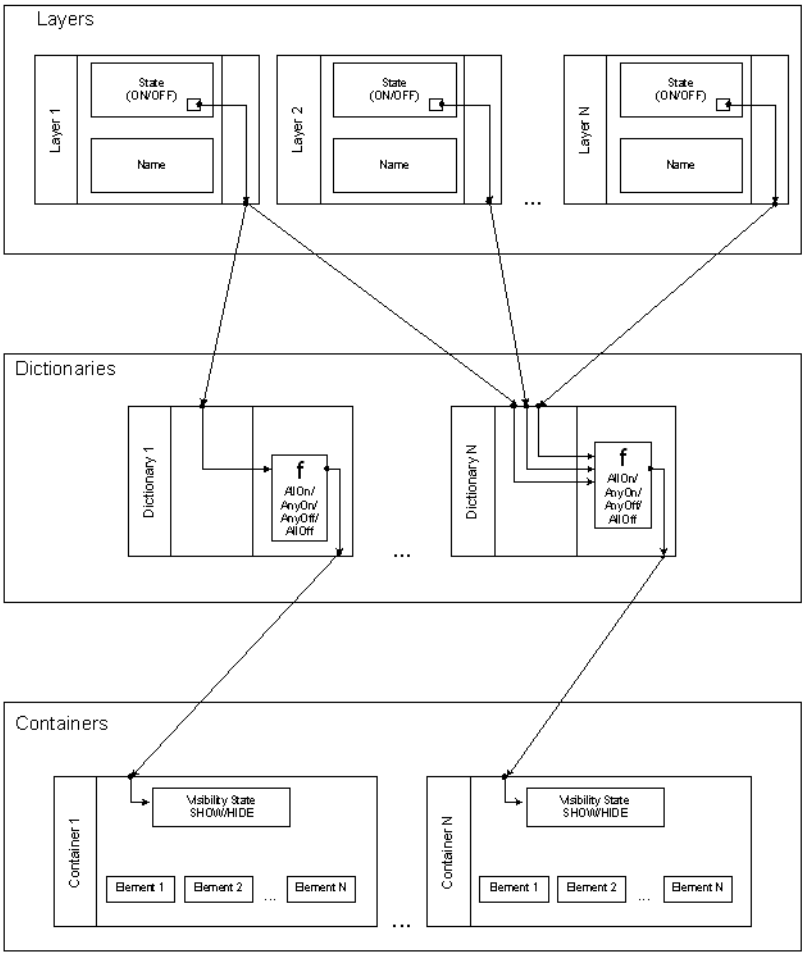
Each container is an arbitrary set of PDF elements or other containers. Each container is associated with a dictionary. The container and all its elements (including the other containers) are displayed when its dictionary's visibility policy Boolean function results with ON.

The following diagram shows the relationship between containers and the other PDF document objects:



Layers Diagram

The following diagram shows interconnection between PDF objects and layers:



Visibility Policy

Dictionary Boolean functions define the visibility policy. It can be one of the following:

- AllOn – equivalent to “AND” function for all input parameters,
- AnyOn – equivalent to “OR” function for all input parameters,
- AnyOff – equivalent to “NOT-OR” function for all input parameters,
- AllOff – equivalent to “NOT-AND” function for all input parameters,

In other words:

AllOn	All the containers from the dictionary will be displayed when all the input layers' states are 'ON'
AnyOn	All the containers from the dictionary will be displayed when at least one of the input layers state is 'ON'
AnyOff	All the containers from the dictionary will be displayed when at least one of the input layers state is 'OFF'
AllOff	All the containers from the dictionary will be displayed when all the input layers' states are 'OFF'

Working with Containers, Dictionaries and Layers using ImageGear PDF sample

Before creating a container you need to create one or more layers using the View -> Layers menu. Make sure to create some simple elements like text, paths or images as well.

In order to create container right click on the PDF page and select Create -> Container. You will be prompted to select all the layers you want to associate with the containers dictionary, which will affect the container's visibility. You also need to select the visibility policy and check all the elements you want to include into the new container. When container is created a new dictionary is automatically created as well.

To show or hide the container's elements use View -> Layers menu to change the layer's state to ON or OFF.

In order to rearrange the container's elements you can right click to delete the existing container, which will free its elements and create the new one.

Distributing PDF and PS Fonts with Your Application

ImageGear toolkit comes with multiple PDF and PS fonts that can be used for developing an application based on ImageGear. However only four fonts are pre-licensed for distribution:

- The two MultiMaster PDF fonts, **AdobeSansMM** (zx_____.mmm) and **AdobeSerifMM** (zy_____.mmm).
- The two PostScript fonts, **Courier** and **NotDefFont**.

These fonts are the property of Adobe® Systems and are fully licensed for distribution with your application.

If you need to distribute additional fonts, you need to get a license for the redistribution of those fonts. However, you are not limited to the fonts supplied with ImageGear; it's possible to add other fonts to be used with ImageGear PDF/PS support. To do this, place your fonts to the PDF "Resource" directory as follows:

- CID fonts (if any) to `...\Resource\PDF\CIDFont`

- CMaps fonts to `...\Resource\PDF\CMap`
- PDF fonts to `...\Resource\PDF\Font`
- PostScript fonts to `...\Resource\PS\Fonts`

Please also note that the fonts that are pre-licensed for distribution (i.e., AdobeSansMM, AdobeSerifMM, "Courier" and "NotDefFont") provide the substitution capabilities for the other fonts, so they are likely to be enough for many cases. The below paragraph provides more technical details regarding font substitution.

The multiple master font format is an extension of the Type 1 font format that allows the generation of a wide variety of typeface styles from a single font program. This is accomplished through the presence of various design dimensions in the font. Examples of design dimensions are weight (light to extra-bold) and width (condensed to expanded). Coordinates along these design dimensions (such as the degree of boldness) are specified by numbers. A particular choice of numbers selects an instance of the multiple master font. Adobe® Technical Note #5015, Type 1 Font Format Supplement, describes multiple master fonts in detail.

PDF Format Filter

If you attach the PDF Component to core ImageGear, you can initialize the PDF Format Filter functionality, which can be accessed through the core ImageGear Format Filter API. The PDF Format Filter supports some features of PDF file processing. These features are determined by **`IG_FLTR_`** flags. This Format Filter also allows you to save PDF raster images with different compressions, depending on image bit depth.

Supported Features

The ImageGear PDF Format Filter supports the following features of PDF file processing:

- **`IG_FLTR_DETECTSUPPORT`** - autodetection
- **`IG_FLTR_PAGEREADSUPPORT`** - single page file reading
- **`IG_FLTR_MPAGEREADPSUPPORT`** - multi-page file reading
- **`IG_FLTR_PAGEINSERTSUPPORT`** - single-page file writing
- **`IG_FLTR_MPAGEWRITEPSUPPORT`** - multi-page file writing
- **`IG_FLTR_PAGEDELETESUPPORT`** - page deleting from multi-page files
- **`IG_FLTR_PAGESWAPSUPPORT`** - page swaping in multi-page files

You can detect, read, write, and work with pages in multi-page PDF files using **`IG_fltr_...()`** functions from ImageGear Format Filters API. Please see Working with Format Filters section in this .

Supported Compressions for Images

The following raster image compressions are supported by ImageGear PDF component for read and write:

- **IG_COMPRESSION_NONE** - for 1, 2, 4, 8, 24, 32 bits per pixel (bpp)
- **IG_COMPRESSION_CCITT_G3** - for 1 bpp
- **IG_COMPRESSION_CCITT_G4** - for 1 bpp
- **IG_COMPRESSION_CCITT_G32D** - for 1 bpp
- **IG_COMPRESSION_RLE** - for 1, 2, 4, 8, 24, 32 bits per pixel
- **IG_COMPRESSION_DEFLATE** - for 1, 2, 4, 8, 24, 32 bits per pixel
- **IG_COMPRESSION_JPEG** - for 8, 24, 32 bits per pixel
- **IG_COMPRESSION_LZW** - for 1, 2, 4, 8 bits per pixel

NOTE: Once the image is compressed, an optional text encoding scheme can be applied to convert binary data to text format. See the description of the **TEXT_ENCODING** control parameter for details.

PDF Format Filter Control Parameters

ImageGear PDF Format Filter provides you with many control parameters that are optional, but can be used for PDF file processing refinement.

- **DEPTH**
- **INC_REND**
- **INDEPENDENT_PAGESIZE**
- **PAGE_HEIGHT**
- **PAGE_WIDTH**
- **PASSWORD**
- **PRINT_DEPTH**
- **PRINT_RESOLUTION_X**
- **PRINT_RESOLUTION_Y**
- **RESOLUTION_X**
- **RESOLUTION_Y**
- **SAVE_FLAGS**
- **SAVE_MAJOR**
- **SAVE_MINOR**
- **SMOOTH_FLAGS**
- **TEXT_ENCODING**
- **USE_CROP_BOX**

DEPTH

This parameter specifies bit depth in bits per pixels during the PDF document convertion into the raster image. The more is this value, the more quality of the raster image you get and the more memory is required for rasterization. The DIB of the output raster image has this value as a bit count.

Data type:

UINT

Default Value:

24

Valid Values:

1, 8, 24

Example:

```
// Get current value of DEPTH control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "DEPTH", FALSE, NULL, NULL,
(LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the DEPTH control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "DEPTH", (LPVOID)1,
sizeof(UINT));
```

INC_REND

This parameter specifaies whether incremental rendering or rendering at once should be performed.

Data type:

Boolean

Default Value:

False

Valid Values:

True, False

Example:

```
// Get current value of INC_REND control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "INC_REND", FALSE, NULL, NULL,
(LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the INC_REND control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "INC_REND", TRUE,
sizeof(UINT));
```

INDEPENDENT_PAGESIZE

(*Used with PDF write only.*) If this parameter is **FALSE,** when saving a raster image into the PDF document, the width and height of the newly created page is set to the width and height of the previous page in the PDF document. If the previous page does not exist, the width and height are calculated from the image resolution and size as follows:

Width = <width of the raster image> * 72 / <X DPI of the raster image>
Height = < height of the raster image> * 72 / <Y DPI of the raster image>

If the image resolution is not defined, the width and height are set to the width and height of the A4 page, which is 612x792.

Use the following parameters to set the page size when this option is **TRUE**:

- **PAGE_HEIGHT**
- **PAGE_WIDTH**

Data type:

AT_BOOL

Valid Values:

TRUE, **FALSE**

Default Value:

FALSE

Example:

```
// Get current value of INDEPENDENT_PAGESIZE control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "INDEPENDENT_PAGESIZE", FALSE,
NULL, NULL, (LPVOID)bIndependentPagesize,
sizeof(bIndependentPagesize));

// Set new value to INDEPENDENT_PAGESIZE control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "INDEPENDENT_PAGESIZE",
(LPVOID)TRUE, sizeof(AT_BOOL));
```

PAGE_HEIGHT

(*Used with PDF write only.*) This parameter sets the height of the page (in 1/72 inches). If this option is **0**, the height is calculated from the image resolution and size as follows:

Height = < height of the raster image> * 72 / <Y DPI of the raster image>

If the image resolution is not defined, the height is set to the height of an A4 page, which is 792.

This option is not used if **RESOLUTION_X** is **FALSE**.

Data type:

UINT

Valid Values:

Any non-negative value.

Default Value:

0

Example:

```
// Get current value of PAGE_HEIGHT control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "PAGE_HEIGHT", FALSE, NULL,
NULL, (LPVOID)nPageHeight, sizeof(nPageHeight));

// Set new value to PAGE_HEIGHT control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "PAGE_HEIGHT", (LPVOID)792,
sizeof(UINT));
```

PAGE_WIDTH

(*Used with PDF write only.*) This parameter sets the width of the page (in 1/72 inches). If this option is **0**, the width is calculated from the image resolution and size as follows:

Width = <width of the raster image> * 72 / <X DPI of the raster image>

If the image resolution is not defined, the width is set to the width of the A4 page, which is 612.

This option is not used if **RESOLUTION_X** is **FALSE**.

Data type:

UINT

Valid Values:

Any non-negative value.

Default Value:

0

Example:

```
// Get current value of PAGE_WIDTH control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "PAGE_WIDTH", FALSE, NULL,
NULL, (LPVOID)nPageWidth, sizeof(nPageWidth));
```

```
// Set new value to PAGE_WIDTH control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "PAGE_WIDTH", (LPVOID)612,
sizeof(UINT));
```

PASSWORD

This parameter specifies the password string for the password of the protected PDF documents.

Data type:

LPCHAR

Default Value:

“”

Valid Values:

Any

```
Example:
// Get current value of PASSWORD control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "PASSWORD", FALSE, NULL, NULL,
(LPVOID)&szPassword, sizeof(szPassword) );
// Set new value to the PASSWORD control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "PASSWORD",
(LPVOID)szPassword, sizeof(szPassword));
```

PRINT_DEPTH

This parameter specifies bit depth in bits per pixels during the PDF document printing. The more is this value, the more quality of the raster image you get and the more memory is required for printing.

Data type:

UINT

Default Value:

8

Valid Values:

1, 8, 24

Example:

```
// Get current value of PRINT_DEPTH control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "PRINT_DEPTH", FALSE, NULL,
NULL, (LPVOID)nPrintDepth, sizeof(nPrintDepth) );
```

```
// Set new value to the PRINT_DEPTH control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "PRINT_DEPTH", (LPVOID)1,
sizeof(UINT));
```

PRINT_RESOLUTION_X

This parameter specifies the horizontal resolution in dots per inch during the PDF document printing. The more is this value, the more is the quality of the image to be printed.

Data type:

UINT

Default Value:

300

Valid Values:

Any

Example:

```
// Get current value of PRINT_RESOLUTION_X control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "PRINT_RESOLUTION_X", FALSE,
NULL, NULL, (LPVOID)nPrintResolutionX,
sizeof(nPrintResolutionX) );
// Set new value to the PRINT_RESOLUTION_X control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "PRINT_RESOLUTION_X",
(LPVOID)300, sizeof(UINT));
```

PRINT_RESOLUTION_Y

This parameter specifies the vertical resolution in dots per inch during the PDF document printing. The more is this value, the more is the quality of the image to be printed.

Data type:

UINT

Default Value:

300

Valid Values:

Any

Example:

```
// Get current value of PRINT_RESOLUTION_Y control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "PRINT_RESOLUTION_Y", FALSE,
NULL, NULL, (LPVOID)nPrintResolutionY,
sizeof(nPrintResolutionY) );
// Set new value to the PRINT_RESOLUTION_Y control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "PRINT_RESOLUTION_Y",
(LPVOID)300, sizeof(UINT));
```

RESOLUTION_X

This parameter specifies the horizontal resolution in dots per inch during the PDF/PS document conversion into the raster image. The higher this value, the higher-quality raster image you get after rasterization. The DIB of the output raster image has this value as an **X** resolution.

Data type:

UINT

Valid Values:

Any

Default Value:

72

Example:

```
// Get current value of RESOLUTION_X control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "RESOLUTION_X", FALSE, NULL,
NULL, (LPVOID)nResolutionX, sizeof(nResolutionX));

// Set new value to the RESOLUTION_X control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "RESOLUTION_X", (LPVOID)300,
sizeof(UINT));
```

RESOLUTION_Y

This parameter specifies the vertical resolution in dots per inch during the PDF/PS document conversion into the raster image. The higher this value, the higher-quality raster image you get after rasterization. The DIB of the output raster image has this value as an **Y** resolution.

Data type:

UINT

Valid Values:

Any

Default Value:

72

Example:

```
// Get current value of RESOLUTION_Y control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "RESOLUTION_Y", FALSE, NULL,
NULL, (LPVOID)nResolutionY, sizeof(nResolutionY));

// Set new value to the RESOLUTION_Y control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "RESOLUTION_Y", (LPVOID)300,
sizeof(UINT));
```

SAVE_FLAGS

(*Used with PDF write only.*) This parameter specifes an option for saving a PDF file, which allows you to remove unreferenced objects, often reducing file size as well as to write a linearized file for page-served remote (network) access.

Data type:

UINT

Default Value:

IG_PDF_OPTIMIZED

Valid Values:

A bit composition of an OR of the following values:

- **IG_PDF_OPTIMIZED** - perform garbage collection on unreferenced objects
- **IG_PDF_LINEARIZED** - writes the file linearized for page serving over the remote connections

Example:

```
// Get current value of SAVE_FLAGS control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "SAVE_FLAGS", FALSE, NULL,
NULL, (LPVOID)nSaveFlags, sizeof(nSaveFlags) );

// Set new value to SAVE_FLAGS control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "SAVE_FLAGS", (LPVOID)
IG_PDF_OPTIMIZED|IG_PDF_LINEARIZED, sizeof(UINT) );
```

SAVE_MAJOR

Specifies major PDF version number of the document for saving. If major equals 0, both major and minor are ignored and the document is saved to the library's default version.

Make sure that the document conforms to the version number you are setting.

Data type:

UINT

Default Value:

0

Valid Values:

1

SAVE_MINOR

Specifies minor PDF version number of the document for saving. Make sure that the document conforms to the version number you are setting.

Data type:

UINT

Default Value:

0

Valid Values:

6, 5, 4 etc.

SMOOTH_FLAGS

This parameter specifes smooth settings for PDF rasterization.

Data type:

UINT

Default Value:

0

Valid Values:

A bit composition of an OR of the following values:

- 1 - Draw smooth text
- 2 - Draw smooth line art
- 4 - Draw smooth image

Example:

```
// Get current value of SMOOTH_FLAGS control parameter
IG_fltr_ctrl_get( IG_FORMAT_PDF, "SMOOTH_FLAGS", FALSE, NULL,
NULL, (LPVOID)nSmoothFlags, sizeof(nSmoothFlags) );
// Set new value to SMOOTH_FLAGS control parameter
IG_fltr_ctrl_set( IG_FORMAT_PDF, "SMOOTH_FLAGS", (LPVOID) 1|4,
sizeof(UINT) );
```

TEXT_ENCODING

(*Used with PDF write only.*) This parameter specifies which encoding scheme should be used to convert binary image data to the text format when saving raster image into the PDF document.

Data type:

UINT

Valid Values:

- **IG_PDF_TEXTENC_NONE** – no encoding used
- **IG_PDF_TEXTENC_ASCII_85** – ASCII 85 encoding used
- **IG_PDF_TEXTENC_ASCII_HEX** – ASCII HEX encoding used

Default Value:

IG_PDF_TEXTENC_ASCII_85

Example:

```
// Get current value of TEXT_ENCODING control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "TEXT_ENCODING", FALSE, NULL,
NULL, (LPVOID)nTextEncoding, sizeof(nTextEncoding));

// Set new value to TEXT_ENCODING control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "TEXT_ENCODING",
(LPVOID)IG_PDF_TEXTENC_NONE, sizeof(UINT));
```

USE_CROP_BOX

TRUE - use PDF crop box rectangle for page layout, FALSE - use PDF media rectangle for page layout.

Data type:

AT_BOOL

Valid Values:

True, False

Default Value:

True

Example:

```
// Get current value of USE_CROP_BOX control parameter
IG_fltr_ctrl_get(IG_FORMAT_PDF, "USE_CROP_BOX", FALSE, NULL,
NULL, (LPVOID)bUseCropBox, sizeof(bUseCropBox));

// Set new value to USE_CROP_BOX control parameter
IG_fltr_ctrl_set(IG_FORMAT_PDF, "USE_CROP_BOX", (LPVOID)FALSE,
sizeof(UINT));
```

PostScript Filter Control Parameters

ImageGear PostScript Filter provides you with many control parameters that are optional but can be used for PostScript file processing refinement.

- **AUTO_POSITION**
- **DEPTH**
- **HEIGHT**
- **INDEPENDENT_PAGESIZE**
- **JOB_OPTIONS_FILE**
- **LEVEL**
- **PAGE_HEIGHT**
- **PAGE_WIDTH**
- **READ_PREVIEW**
- **RESOLUTION_X**
- **RESOLUTION_Y**
- **SMOOTH_FLAGS**
- **TEXT_ENCODING**
- **TYPE**
- **USE_CROP_BOX**
- **WIDTH**

AUTO_POSITION

This parameter enables/disables `AutoPositionEPSFiles` option for reading PS/EPS file format.

Data type:

Boolean

Default Value:

TRUE

Valid Values:

TRUE, FALSE

Example:

```
// Get current value of AUTO_POSITION control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "AUTO_POSITION", FALSE,
NULL, NULL, (LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the AUTO_POSITION control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "AUTO_POSITION",
(LPVOID)FALSE, sizeof(UINT));
```

DEPTH

This parameter specifies bit depth in bits per pixels during the PostScript document convertion into the raster image. The more is this value, the more quality of the raster image you get and the more memory is required for the rasterization. The DIB of the output raster image has this value as a bit count.

Data type:

UINT

Default Value:

24

Valid Values:

1, 8, 24

Example:

```
// Get current value of DEPTH control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "DEPTH", FALSE, NULL,
NULL, (LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the DEPTH control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "DEPTH", (LPVOID)1,
sizeof(UINT));
```

HEIGHT

This parameter determines the height of PS page to load.

Data type:

UINT

Default Value:

0

Valid Values:

Any positive value

Example:

```
// Get current value of HEIGHT control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "HEIGHT", FALSE, NULL,
NULL, (LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the HEIGHT control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "HEIGHT", (LPVOID)1,
sizeof(UINT));
```

INDEPENDENT_PAGESIZE

(*Used with PostScript write only.*) When this parameter is **FALSE,** when saving a raster image into the PostScript document, the width and height of the newly created page is set to the width and height of the previous page in the PostScript document. If the previous page does not exist, the width and height are calculated from the image resolution and size as follows:

Width = <width of the raster image> * 72 / <X DPI of the raster image>

Height = < height of the raster image> * 72 / <Y DPI of the raster image>

If the image resolution is not defined, the width and height are set to the width and height of the A4 page, which is 612x792.

Use the following parameters to set the page size when this option is **TRUE**:

- **PAGE_HEIGHT**
- **PAGE_WIDTH**

Data type:

AT_BOOL

Valid Values:

TRUE, **FALSE**

Default Value:

FALSE

Example:

```
// Get current value of INDEPENDENT_PAGESIZE control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "INDEPENDENT_PAGESIZE",
FALSE, NULL, NULL, (LPVOID)bIndependentPagesize,
sizeof(bIndependentPagesize));

// Set new value to INDEPENDENT_PAGESIZE control parameter
```

```
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "INDEPENDENT_PAGESIZE",
(LPVOID)TRUE, sizeof(AT_BOOL));
```

JOB_OPTIONS_FILE

This parameter determines JobOptions file name.

Data type:

LPSTR

Default Value:

N/A

Valid Values:

Any string

Example:

```
// Get current value of JOB_OPTIONS_FILE control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "JOB_OPTIONS_FILE",
FALSE, NULL, NULL, (LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the JOB_OPTIONS_FILE control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "JOB_OPTIONS_FILE",
(LPVOID)JOBOPT_1, sizeof(UINT));
```

LEVEL

(Used with PostScript write only.) This parameter specifies a language level of the output PostScript 3.0 document when saving to PostScript.

Data type:

UINT

Valid Values:

- **IG_PS_LEVEL_1** - language level 1
- **IG_PS_LEVEL_2** - language level 2
- **IG_PS_LEVEL_3** - language level 3

Default Value:

IG_PS_LEVEL_3

Example:

```
// Get current value of LEVEL control parameter
```

```
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, " LEVEL ", FALSE, NULL,
NULL, (LPVOID)nLevel, sizeof(nLevel) );

// Set new value to LEVEL control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "LEVEL", (LPVOID)
IG_PS_LEVEL_2, sizeof(UINT) );
```

PAGE_HEIGHT

(*Used with PostScript write only.*) This parameter sets the height of the page (in 1/72 inches). If this option is **0**, the height is calculated from the image resolution and size as follows:

Height = < height of the raster image> * 72 / <Y DPI of the raster image>

If the image resolution is not defined, the height is set to the height of an A4 page, which is 792.

This option is not used if **RESOLUTION_X** is **FALSE**.

Data type:

UINT

Valid Values:

Any non-negative value.

Default Value:

0

Example:

```
// Get current value of PAGE_HEIGHT control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "PAGE_HEIGHT", FALSE,
NULL, NULL, (LPVOID)nPageHeight, sizeof(nPageHeight));

// Set new value to PAGE_HEIGHT control parameter
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "PAGE_HEIGHT",
(LPVOID)792, sizeof(UINT));
```

PAGE_WIDTH

(*Used with PostScript write only.*) This parameter sets the width of the page (in 1/72 inches). If this option is **0**, the width is calculated from the image resolution and size as follows:

Width = <width of the raster image> * 72 / <X DPI of the raster image>

If the image resolution is not defined, the width is set to the width of the A4 page, which is 612.

This option is not used if **RESOLUTION_X** is **FALSE**.

Data type:

UINT

Valid Values:

Any non-negative value.

Default Value:

0

Example:

```
// Get current value of PAGE_WIDTH control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "PAGE_WIDTH", FALSE,
NULL, NULL, (LPVOID)nPageWidth, sizeof(nPageWidth));

// Set new value to PAGE_WIDTH control parameter
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "PAGE_WIDTH",
(LPVOID)612, sizeof(UINT));
```

READ_PREVIEW

This parameter determines whether to read TIFF preview only or not.**Data type:**

Boolean

Default Value:

FALSE

Valid Values:

TRUE, FALSE

Example:

```
// Get current value of READ_PREVIEW control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "READ_PREVIEW", FALSE,
NULL, NULL, (LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the READ_PREVIEW control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "READ_PREVIEW",
(LPVOID)TRUE, sizeof(UINT));
```

RESOLUTION_X

This parameter specifies the horizontal resolution in dots per inch during the PostScript/PDF document conversion into the raster image. The higher this value, the higher-quality raster image you get after rasterization. The DIB of the output raster image has this value as an **X** resolution.

Data type:

UINT

Valid Values:

Any

Default Value:

72

Example:

```
// Get current value of RESOLUTION_X control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "RESOLUTION_X", FALSE,
NULL, NULL, (LPVOID)nResolutionX, sizeof(nResolutionX));

// Set new value to the RESOLUTION_X control parameter
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "RESOLUTION_X",
(LPVOID)300, sizeof(UINT));
```

RESOLUTION_Y

This parameter specifies the vertical resolution in dots per inch during the PostScript/PDF document conversion into the raster image. The higher this value, the higher-quality raster image you get after rasterization. The DIB of the output raster image has this value as an **Y** resolution.

Data type:

UINT

Valid Values:

Any

Default Value:

72

Example:

```
// Get current value of RESOLUTION_Y control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "RESOLUTION_Y", FALSE,
NULL, NULL, (LPVOID)nResolutionY, sizeof(nResolutionY));
```

```
// Set new value to the RESOLUTION_Y control parameter
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "RESOLUTION_Y",
(LPVOID)300, sizeof(UINT));
```

SMOOTH_FLAGS

This parameter specifes smooth settings for PostScript rasterization.

Data type:

UINT

Default Value:

0

Valid Values:

A bit composition of an OR of the following values:

- 1 - Draw smooth text
- 2 - Draw smooth line art
- 4 - Draw smooth image

Example:

```
// Get current value of SMOOTH_FLAGS control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "SMOOTH_FLAGS", FALSE,
NULL, NULL, (LPVOID)nSmoothFlags, sizeof(nSmoothFlags) );
// Set new value to SMOOTH_FLAGS control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "SMOOTH_FLAGS",
(LPVOID) 1|4, sizeof(UINT) );
```

TEXT_ENCODING

(*Used with PostScript write only.*) This parameter specifies which encoding scheme should be used to convert binary image data to the text format when saving raster image into the PostScript document.

Data type:

UINT

Valid Values:

- **IG_PDF_TEXTENC_NONE** – no encoding used
- **IG_PDF_TEXTENC_ASCII_85** – ASCII 85 encoding used
- **IG_PDF_TEXTENC_ASCII_HEX** – ASCII HEX encoding used

Default Value:

IG_PDF_TEXTENC_ASCII_85

Example:

```
// Get current value of TEXT_ENCODING control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "TEXT_ENCODING", FALSE,
NULL, NULL, (LPVOID)nTextEncoding, sizeof(nTextEncoding));

// Set new value to TEXT_ENCODING control parameter
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "TEXT_ENCODING",
(LPVOID)IG_PDF_TEXTENC_NONE, sizeof(UINT));
```

TYPE

(*Used with PostScript write only*.) This parameter specifies a type of the output PostScript document when saving to PostScript.

Data type:

UINT

Valid Values:

- **IG_POSTSCRIPT** - PS file
- **IG_EPS_NO_PREVIEW** - EPS file with no preview
- **IG_EPS_STANDARD_PREVIEW** - EPS file with standard preview
- **IG_EPS_EXTENDED_PREVIEW** - EPS file with extended preview

Default Value:

IG_POSTSCRIPT

Example:

```
// Get current value of TYPE control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "TYPE", FALSE, NULL,
NULL, (LPVOID)nType, sizeof(nType) );

// Set new value to TYPE control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "TYPE", (LPVOID)
IG_EPS_STANDARD_PREVIEW, sizeof(UINT) );
```

TRUE - use PDF crop box rectangle for page layout, FALSE - use PDF media rectangle for page layout.

Data type:

AT_BOOL

Valid Values:

True, False

Default Value:

True

Example:

```
// Get current value of USE_CROP_BOX control parameter
IG_fltr_ctrl_get(IG_FORMAT_POSTSCRIPT, "USE_CROP_BOX", FALSE,
NULL, NULL, (LPVOID)bUseCropBox, sizeof(bUseCropBox));

// Set new value to USE_CROP_BOX control parameter
IG_fltr_ctrl_set(IG_FORMAT_POSTSCRIPT, "USE_CROP_BOX",
(LPVOID)FALSE, sizeof(UINT));
```

WIDTH

This parameter determines the width of PS page to load.

Data type:

UINT

Default Value:

0

Valid Values:

Any positive value

Example:

```
// Get current value of WIDTH control parameter
IG_fltr_ctrl_get( IG_FORMAT_POSTSCRIPT, "WIDTH", FALSE, NULL,
NULL, (LPVOID)nDepth, sizeof(nDepth) );
// Set new value to the WIDTH control parameter
IG_fltr_ctrl_set( IG_FORMAT_POSTSCRIPT, "WIDTH", (LPVOID)1,
sizeof(UINT));
```

GS PDF Component

GhostScript-based PDF Component allows you to load, save and process **PDF (Portable Document Format)** file format images using core ImageGear and other IG Components functionality.

NOTE: The Adobe®-based PDF Component is applicable for Linux (i386), Solaris 32-bit, and AIX Unix Platforms.

Linux for PowerPC, HP-UX 32-bit & 64-bit, HP-UX for Itanium, and Sun Solaris 64-bit use the GS PDF Component .

This Component also allows to read vector data from **PS (PostScript)** and **EPS (Encapsulated PostScript)** file formats (core ImageGear Component allows to read only raster data from these two formats).

Besides this PDF Component provides you with possibility of text extraction from loaded PDF and PS image file using function **`IG_PDF_text_extract()`**.

NOTE: The ImageGear PDF Component performs 'rasterization' during the loading of a PDF file, i.e. it converts all PDF objects from the file (including vector data) to a bitmap. When IG saves the image as a PDF, the resulting PDF file does not contain data in a vector form but rather consists of 1 raster object; as a result, vector oriented PDF viewers such as Adobe Acrobat cannot provide some of the benefits when reading PDF saved by ImageGear, e.g. the ability to search for a specific text inside a file. You can use ImageGear to create new PDF images and to add, delete, or swap pages in existing PDF files that were created using ImageGear. However, you cannot add, delete, or swap pages in existing PDF files that were created using other tools.

PDF support is compatible with PDF 1.0, 1.1, 1.2, and 1.3 as defined in the Portable Document Format Reference Manual Version 1.3 of March 11, 1999, distributed by Adobe Systems Incorporated, except as noted with the following exceptions:

- Masked images.
- Type 2 (smooth shading) Pattern color spaces, although the sh operator for smooth-shaded fills is implemented.
- Native Separation and DeviceN color spaces -- the alternate space is always used.
- Native ICCBased color spaces -- these too always use the alternate space.
- In-RIP trapping.

PS/EPS support is compatible with the PostScript (TM) language with the following exceptions:

- Native Separation and DeviceN color spaces - the alternate space is always used.
- Settable ProcessColorModel for page devices, except for a very few special devices.
- IODevices other than %stdin, %stdout, %stderr, %lineedit, %statementedit,

%os%, and (if configured) %pipe%.

- In-RIP trapping.

How to Attach PDF Component to Core ImageGear

To use the ImageGear PDF Component (for HP-UX platform), you have to attach the component to core ImageGear using the following function call:

```
IG_comm_comp_attach("PDF")
```

To check if the Component is attached successfully use the function:

```
IG_comm_comp_check("PDF")
```

This function (method) returns **TRUE** if the Component is attached.

Retrieving Info About the Component

To retrieve information about attached Component call the function:

```
IG_comm_comp_list(LPUINT *lpnCount, UINT nIndex, LPCHAR lpComp,
DWORD dwCompSize, LPUINT lpnRevMajor, LPUINT lpnRevMinor, LPUINT
lpnRevUpdate,
LPCHAR lpBuildDate, UINT nBDSize, LPCHAR lpInfoStr, UINT
nISSize)
```

This function provides you with the full list of info about the component determined by **nIndex** index from the list of currently attached components whose number are returned through **lpnCount** argument.

For more detailed information about these functions' usage see Using ImageGear Component Manager section in this Chapter.

PDF Format Filter (for HP-UX)

If you attach the PDF Component to core ImageGear, it allows you to initialize PDF Format Filter functionality, which can be accessed through core ImageGear Format Filter API. PDF Format Filter supports some features of PDF file processing. These features are determined by **IG_FLTR_** flags. This Format Filter also allows you to save PDF raster images with different compressions, depending of image bit depth.

Supported Features

ImageGear PDF Format Filter supports the following features of PDF file processing:

- **IG_FLTR_DETECTSUPPORT** - autodetection.
- **IG_FLTR_PAGEREADSUPPORT** - single page file reading.
- **IG_FLTR_MPAGEREADPSUPPORT** - multi-page file reading.
- **IG_FLTR_PAGEINSERTSUPPORT** - single-page file writing.
- **IG_FLTR_MPAGEWRITEPSUPPORT** - multipage file writing.
- **IG_FLTR_PAGEDELETESUPPORT** - page deleting from multi-page file.
- **IG_FLTR_PAGESWAPSUPPORT** - page swaping in multi-page files.

You can detect, read, write and work with pages in multi-page PDF files using **IG_fltr_...()** functions from ImageGear Format Filters API. Please see Working with Format Filters section in this Chapter.

Supported Compressions for Images

ImageGear PDF Format Filter allows you to save PDF image file using the following compressions:

- **IG_COMPRESSION_NONE** - for 1, 4, 8, 24 bpp
- **IG_COMPRESSION_CCITT_G3** - for 1 bpp
- **IG_COMPRESSION_CCITT_G4** - for 1 bpp
- **IG_COMPRESSION_CCITT_G32D** - for 1 bpp
- **IG_COMPRESSION_JPEG** - for 8, 24 bpp

NOTE: Once the image is compressed, an optional text encoding scheme can be applied to convert binary data to text format. See the description of the **TEXT_ENCODING** control parameter for details.

You can save PDF files using ImageGear functions **IG_save_file()** or **IG_fltr_save_file()** and **SAVE** constants:

- **IG_SAVE_PDF_UNCOMP**
- **IG_SAVE_PDF_G3**
- **IG_SAVE_PDF_G3_2D**
- **IG_SAVE_PDF_G4**
- **IG_SAVE_PDF_JPG**

PDF Format Filter Control Parameters (for HP-UX)

ImageGear PDF Format Filter provides you the following optional control parameters for PDF file processing perfection:

- **DEPTH**
- **FILE_NAME**
- **GRAPHICS_ALPHA**
- **INDEPENDENT_PAGESIZE**
- **NOCIE**
- **NOTRANSPARENCY**
- **PAGE_HEIGHT**
- **PAGE_WIDTH**
- **RESOLUTION_X**
- **RESOLUTION_Y**
- **TEXT_ALPHA**
- **TEXT_ENCODING**

- **USE_CROP_BOX**

See the reference of these parameters below.

DEPTH

Data type:

UINT

Purpose:

Sets the page bitmap depth in bits per pixels for the display.

Valid Values:

1, 4, 8, 24

Default Value:

1

NOTE: The default selects the highest bit depth supported by your display driver. You should not set this higher than your actual display depth because doing so uses extra memory for the page bitmap, and will not improve the display.

Example:

```
/* get current value of DEPTH control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "DEPTH", FALSE, NULL, NULL,
(LPVOID)nDepth, sizeof(nDepth));

/* set new value to DEPTH control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "DEPTH", (LPVOID)8,
sizeof(nDepth));
```

FILE_NAME

Data type:

CHAR [262]

Purpose:

Specifies the name of the PDF or PS/EPS file to be loaded.

Valid Values:

Any valid path to a PDF or PS/EPS file.

Default Value:

" "

Example:

```
/* get current value of FILE_NAME control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "FILE_NAME", FALSE, NULL, NULL,
szFileName, sizeof(szFileName));

/* set new value to FILE_NAME control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "FILE_NAME",
(LPVOID)"c:\example.pdf", sizeof(szFileName));
```

GRAPHICS_ALPHA

Data type:

UINT

Purpose:

Sets anti-aliasing (smoothing) for graphics and for text that is too large to fit in the font cache.

Valid Values:

1, 2, 4

Default Value:

1

NOTE: Using anti-aliasing slows down the image drawing. **GRAPHICS_ALPHA** can only be used if your display depth is set to 8 bits per pixel or higher.

Example:

```
/* get current value of GRAPHICS_ALPHA control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "GRAPHICS_ALPHA", FALSE, NULL,
NULL, (LPVOID)nGraphicsAlpha, sizeof(nGraphicsAlpha));

/* set new value to GRAPHICS_ALPHA control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "GRAPHICS_ALPHA", (LPVOID)2,
sizeof(nGraphicsAlpha));
```

INDEPENDENT_PAGESIZE

Data type:

BOOL

Purpose:

When this parameter is **FALSE,** the width and height of the page is set to the width and height of the previous page in the multi-page image. (Used with PDF write.)

Use the following parameters to set the page size when this option is **TRUE**:

- **PAGE_HEIGHT**
- **PAGE_WIDTH**

Valid Values:

TRUE, FALSE

Default Value:

FALSE

Example:

```
/* get current value of INDEPENDENT_PAGESIZE control parameter
*/
IG_fltr_ctrl_get(IG_FORMAT_PDF, "INDEPENDENT_PAGESIZE", FALSE,
NULL, NULL, (LPVOID)bIndependentPagesize,
sizeof(bIndependentPagesize));

/* set new value to INDEPENDENT_PAGESIZE control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "INDEPENDENT_PAGESIZE",
(LPVOID)TRUE, sizeof(AT_BOOL));
```

NOCIE

Data type:

BOOL

Purpose:

This parameter, when set to **TRUE**, disables the support for ICC input profiles from a PDF file. Therefore it allows to speed up the PDF file loading, if the image has ICC input profiles.

Valid Values:

TRUE, FALSE

Default Value:

FALSE

Example:

```
/* get current value of NOCIE control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "NOCIE", FALSE, NULL, NULL,
(LPVOID)bNocie, sizeof(bNocie));

/* set new value to NOCIE control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "NOCIE", (LPVOID)TRUE,
sizeof(AT_BOOL));
```

NOTRANSPARENCY

Data type:

BOOL

Purpose:

This parameter, when set to **TRUE**, disables the transparency rendering of a PDF file. Therefore it allows to speed up the PDF file loading, if the image has transparent objects.

Valid Values:

TRUE, FALSE

Default Value:

FALSE

Example:

```
/* get current value of NOTRANSPARENCY control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "NOTRANSPARENCY", FALSE, NULL,
NULL,
(LPVOID)bNotransparency, sizeof(bNotransparency));

/* set new value to NOTRANSPARENCY control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "NOTRANSPARENCY", (LPVOID)TRUE,
sizeof(AT_BOOL));
```

PAGE_HEIGHT

Data type:

UINT

Purpose:

This parameter sets the height of the page (in 1/72 inches). If it is **0**, the height is calculated from the image resolution and size. If the image resolution is not defined, the height is set to the height of an A4 page.

This option is not used if **RESOLUTION_X** is **FALSE**.

(Used with PDF write)

Valid Values:

Any non-negative value.

Default Value:

0

Example:

```
/* get current value of PAGE_HEIGHT control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "PAGE_HEIGHT", FALSE, NULL,
NULL, (LPVOID)nPageHeight, sizeof(nPageHeight));

/* set new value to PAGE_HEIGHT control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "PAGE_HEIGHT", (LPVOID)720,
sizeof(nPageHeight));
```

PAGE_WIDTH

Data type:

UINT

Purpose:

This parameter sets the width of the page (in 1/72 inches). If this option is **0**, the width is calculated from the image resolution and size. If the image resolution is not defined, the width is set to the width of the A4 page.

This option is not used if **RESOLUTION_X** is **FALSE**.

(Used with PDF write)

Valid Values:

Any non-negative value.

Default Value:

0

Example:

```
/* get current value of PAGE_WIDTH control parameter */
```

```
IG_fltr_ctrl_get(IG_FORMAT_PDF, "PAGE_WIDTH", FALSE, NULL,
NULL, (LPVOID)nPageWidth, sizeof(nPageWidth));

/* set new value to PAGE_WIDTH control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "PAGE_WIDTH", (LPVOID)720,
sizeof(nPageWidth));
```

RESOLUTION_X

Data type:

UINT

Purpose:

Sets the display horizontal resolution in dots per inch.

Valid Values:

Depends on the amount of available memory. Typically 96 – 300, but you may specify any number.

Default Value:

96

NOTE: The default for a VGA display is 96 dots per inch.

Example:

```
/* get current value of RESOLUTION_X control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "RESOLUTION_X", FALSE, NULL,
NULL, (LPVOID)nResolutionX, sizeof(nResolutionX));

/* set new value to RESOLUTION_X control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "RESOLUTION_X", (LPVOID)300,
sizeof(nResolutionX));
```

RESOLUTION_Y

Data type:

UINT

Purpose:

Sets the display vertical resolution in dots per inch.

Valid Values:

Depends on the amount of available memory. Typically 96 – 300, but you may specify any number.

Default Value:

96

NOTE: **NOTE:** The default for a VGA display is 96 dots per inch.

Example:

```
/* get current value of RESOLUTION_Y control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "RESOLUTION_Y", FALSE, NULL,
NULL, (LPVOID)nResolutionY, sizeof(nResolutionY));

/* set new value to RESOLUTION_Y control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "RESOLUTION_Y", (LPVOID)300,
sizeof(nResolutionY));
```

TEXT_ALPHA

Data type:

UINT

Purpose:

Sets anti-aliasing (smoothing) for fonts. The default (no anti-aliasing) is 1 bit. To enable anti-aliasing of fonts, use 2 bits or 4 bits.

Valid Values:

1, 2, 4

Default Value:

1

NOTE: Using anti-aliasing slows down drawing. **TEXT_ALPHA** can only be used if your display depth is set to 8 bits per pixel or higher.

Example:

```
/* get current value of TEXT_ALPHA control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "TEXT_ALPHA", FALSE, NULL,
NULL, (LPVOID)nTextAlpha, sizeof(nTextAlpha));
```

```
/* set new value to TEXT_ALPHA control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "TEXT_ALPHA", (LPVOID)2,
sizeof(nTextAlpha));
```

TEXT_ENCODING

Data type:

UINT

Purpose:

Specifies which encoding scheme should be used to convert binary image data to text format. (Used with PDF write.)

Valid Values:

- **IG_PDF_TEXTENC_NONE** – no encoding used
- **IG_PDF_TEXTENC_ASCII_85** – ASCII 85 encoding used
- **IG_PDF_TEXTENC_ASCII_HEX** – ASCII HEX encoding used

Default Value:

IG_PDF_TEXTENC_ASCII_85

Example:

```
/* get current value of TEXT_ENCODING control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "TEXT_ENCODING", FALSE, NULL,
NULL, (LPVOID)nTextEncoding, sizeof(nTextEncoding));

/* set new value to TEXT_ENCODING control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "TEXT_ENCODING",
(LPVOID)IG_PDF_TEXTENC_NONE, sizeof(nTextEncoding));
```

USE_CROP_BOX

If this parameter is set to TRUE and PDF page is loaded, the page content is clipped (cropped) to the CropBox rectangle, specified in the PDF page.

When it is set to FALSE, the MediaBox rectangle is used to define the page's boundaries.

- CropBox - A rectangle, defining the visible region of default user space.
- MediaBox - A rectangle, defining the boundaries of the physical medium on which the page is intended to be displayed or printed.

Data type:

BOOL

Valid Values:

TRUE, FALSE

Default Value:

FALSE

Example:

```
/* get current value of USE_CROP_BOX control parameter */
IG_fltr_ctrl_get(IG_FORMAT_PDF, "USE_CROP_BOX", FALSE, NULL,
NULL, (LPVOID)bUse_crop_box, sizeof(bUse_crop_box));

/* set new value to USE_CROP_BOX control parameter */
IG_fltr_ctrl_set(IG_FORMAT_PDF, "USE_CROP_BOX", (LPVOID)TRUE,
sizeof(AT_BOOL));
```

Extracting PostScript and PDF Text (for HP-UX)

ImageGear's PDF Component allows you to extract text from Portable Data Format (PDF) or PostScript documents using **IG_PDF_text_extract**. This ability is useful for finding certain words in the documents, for retrieving information, and for recovering text from Postscript or PDF files when the source is lost.

ImageGear can read the original PDF file and extract the textual content at just about 100% accuracy (English text is approximated at close to 100%, but some symbols in the extracted text, such as "-", may be approximated incorrectly). This is not OCR output; rather the software scans the original source document (a PDF file), ignores all the format characters and notation and outputs all the words.

The text extraction routine extracts words in the same sequence as they are rendered by the document. This usually, but not always, follows the order that a human would read the words on a page. Within this sequence, words are separated by either space or new line depending on whether or not they fall on the same line. Each page is terminated with a formfeed.

IG_PDF_text_extract

You can call PDF component text extracting function using two ways. Firstly, you may call this function through the macro defined in **i_PDF.h** public header file:

```
#define IG_PDF_text_extract(_lpszFileName, _lpszTextName,
_nPage, _nPageCount)((AT_ERRCOUNT (CACCUAPI *)( LPCHAR,LPSTR,
LPSTR, UINT, UINT))
IG_comm_function_call)("PDF.IG_PDF_text_extract",
(_lpszFileName), (_lpszTextName), (_nPage), (_nPageCount))
```

In this case your application will search all included public headers for this macro and call this function through **IG_comm_function_call()** component manager function call that is also determined in **i_PDF.h**. So if you are going to use PDF text extracting function in a multiple loop we would recommend you the second way of using this function type declaration that is also determined in **i_PDF.h** file:

```
typedef  AT_ERRCOUNT (LPACCUAPI LPAFT_PDF_TEXT_EXTRACT)(
   LPCHAR   lpFileName,
   LPCHAR   lpTextFileName,
   UINT           nStartPage,
   UINT           nPageCount
```

```
);
```

So you should declare the variable of this function type and then use **`IG_comm_entry_request()`** to initialize this variable with correct value and call it then.

For more detailed information on how to call ImageGear component function, please see the section Component Manager API from this Chapter.

Arguments:

lpFileName	The name of the PDF or PS document
lpTextFileName	The name of the output TXT file
nStartPage	The number of the first page
nPageCount	The total number of pages to be processed (starting at **`nPage`**)

Description:

This function extracts text from pages determined by arguments **`nStartPage`** and **`nPageCount`**. It extracts text from **`nStartPage`** to (**`nStartPage + nPageCount-1)`**. It is necessary to include the "**`i_PDF.h`**" public header file, which provides the declaration of this function.

Example:

```
#include "i_PDF.h"
…
IG_PDF_text_extract( lpszMyDocument, lpszOutputText, 1, 10)
```

Distributing Applications with the PDF Component

To distribute the application built with the ImageGear PDF component, you must set up the end-user's machine with the appropriate directories and files so that the component performs properly.

It is necessary to install additional files for the PDF extension. Because the ImageGear for Unix PDF and PostScript support implementation is based on Artifex Ghostscript, you must install the files necessary to run the Unix version of GhostScript.

The subdirectory **$HOME/accusoft/accugs** contains the files necessary to run the Unix version of Ghostscript. Either:

Set up the environment variable **IMAGE_GEAR_ACCUGS** to point to the **IMAGE_GEAR_ACCUGS=$HOME/accusoft/.../accugs** directory

or,

If this variable is not defined, ImageGear searches the **/accugs** directory with all files in the subdirectory **/usr/loca**l. Copy **$HOME/accusoft/.../accugs** to **/usr/loca**l.

Creating Your Imaging Application

The previous chapter, Using ImageGear, provided information about how to define and reference ImageGear's data types, structures, and constants, how to call ImageGear functions, and how to detect and handle errors in order to debug your application.

This chapter shows you how to compile and link your program, and discusses the final steps in preparing your application for the end user. Because all the imaging functionality that you must incorporate is contained in a single shared library, this is actually quite easy.

See Also:

- Compiling and Linking
- Preparing Your Application for the End User

Compiling and Linking

UNIX-based applications that call ImageGear functions may be compiled using any ANSI C or C++ compiler though some compilers' switch settings may be slightly different from those shown in the examples below.

To compile and link your program, you must:

1. Inform the compiler where to find the standard and ImageGear "#include" files
2. Inform the linker where to find the ImageGear library
3. Inform the linker which libraries to use

To inform the compiler of the path to your "#include" files, use the `-I` switch. If `gear.h` is in `$HOME/AccuSoft/ImageGear15-64/Pro/Unix/Include` directory, use the following specification

```
-I$HOME/AccuSoft/ImageGear15-64/Pro/Unix/Include
```

Use a switch such as `-L` to direct the linker to the ImageGear library file, using:

```
-L$HOME/AccuSoft/ImageGear15-64/Pro/Unix/Bin/<platform>
```

Use a switch such as -l, to instruct the linker to the libraries to be included in the link process. You must include the ImageGear library:

```
-lIGCORE15
```

To inform the compiler of the path to your "#include" files, use the **`-I`** switch. If `gear.h` is in `$HOME/AccuSoft/ImageGear15-64/Professional Edition/Unix/Include` directory, use the following specification

```
-I/home/accusoft/imagegear15-64/professional edition/unix/
include/gear
```

Use a switch such as **-L** to direct the linker to the ImageGear library file, using:

```
-L/home/accusoft/imagegear15-64/professional edition/unix/bin/
<platform>/gear
```

Use a switch such as `-l`, to instruct the linker to the libraries to be included in the link process. You must include the ImageGear library:

```
-lImageGear
```

See Also:

- Toolkit Libraries
- X-Windows System and Math Library
- Compilation Flags

Toolkit Libraries

You may need to include toolkit libraries using:

```
-lXm -lXt
```

X-Windows System and Math Library

In addition to linking with `libIGCORE15.a`, you must also link with the X Windows System library (`libX11.a`), the math library (libm.a), and the thread support library (`libpthread`). These libraries should be in the following order:

```
-lIGCORE15 -lX11 -lm -lpthread
```

Depending on the platform, the dynamic loader also may be required:

```
-l IGCORE15 -lX11 -lm -lpthread –ldl
```

In addition to linking with `libImageGear.a`, you must also link with the X Windows System library (`libX11.a`), the math library (`libm.a`), and the thread support library (`libpthread`). These libraries should be in the following order:

```
-lImageGear  -lX11 -lm -lpthread
```

Depending on the platform, the dynamic loader also may be required:

```
-lImageGear -lX11 -lm -lpthread -ldl
```

Compilation Flags

You must also define certain flags compiler. Most, if not all, will be included by the preprocessor, but you may find it necessary to define them yourself. They are:

```
-D_UNIX64 (all platforms)
-D__unix (all platforms)
-D__sun (Solaris)
-D__hpux (HP)
```

The `compile/link` will build your application's Executable (`.EXE`) file. If you link to the shared ImageGear library, the path to that library must be included in the library path environment variable, such as `LD_LIBRARY_PATH` (Sun) or `SHLIB_PATH` (HP). That directs the system to the shared library files.

You must also define certain flags compiler. Most, if not all, will be included by the preprocessor, but you may find it necessary to define them yourself. They are:

```
-D__unix (all platforms)
-D__sun (Solaris)
-D__aix (AIX)
-D__hpux (HP)
```

The compile/link will build your application's Executable (.EXE) file. If you link to the shared ImageGear library, the path to that library must be included in the library path environment variable, such as `LD_LIBRARY_PATH` (Sun) or `SHLIB_PATH` (HP). That directs the system to the shared library files.

Preparing Your Application for the End User

To prepare your application for the end-user, complete the following steps:

1. Remove your debugging error messages

   The section Detecting and Handling Errors describes how to produce ImageGear Error Reports reports in a Message Box window to debug your application. Once your application is debugged, however, and you are in the final steps of preparing your application for the end user, you should remove the error messages before distributing the application. In the finalized application to be distributed, all error messages that display should be meaningful to the intended user.

2. Exclude files licensed only for your use

   Of the files you have received in your ImageGear Toolkit, you are authorized to distribute only the ImageGear shared library to the end user. Remove any other AccuSoft ImageGear files from your final distribution, including source code and executables, that you may have included in your application project while developing it.

   Note however that you are permitted to include in developing your application, as you wish, source code from the sample source programs provided to you in directory `$HOME/AccuSoft/ImageGear15-64/Pro/Unix/Samples` `$HOME/AccuSoft/ImageGear15/Professional Edition/Unix/Samples` and its subdirectories. However, you are not permitted to distribute the sample programs themselves.

3. Provide the finalized application's link to the Shared Library

   If your product uses the ImageGear shared library, make sure that after installing your product on the end user's system, you inform the end user to add the path to the ImageGear library to their library path environment variable.

If the shared library cannot be found, an error will occur when an attempt is made to load it.

File Format Reference

This chapter provides information about the image formats that ImageGear supports. Before you begin, please refer to the Encoding vs. Compressing section below to familiarize yourself with the terms used in this Chapter.

Then you can read the Format Suitability at a Glance section, which briefly delineates which formats best support various types of images. This section provides you with a starting point for deciding which file formats to use in your application.

The section ImageGear Support for Graphics File Formats describes the types of imaging file formats supported by ImageGear and provides useful information about support for some specific formats:

- Support for Adobe PDF/PS Formats
- Support for CAD/Vector Formats
- Support for DICOM File Format
- Support for JBIG2 File Format
- Support for JPEG2000 File Formats
- Support for Metafile Formats
- Support for Multimedia File Formats
- Support for Multi-page File Formats

ImageGear Supported Bit Depths section describes the bit depths, and the read/write capabilities of the supported formats. Using this table you can easily find out whether an image can be converted to a particular format.

Detailed information about every ImageGear supported imaging file format or compression can be found in the following sections:

- ImageGear Supported Compressions Reference - here you will find information for every ImageGear supported imaging compression.
- ImageGear Supported File Formats Reference - provides you with the detailed information about every ImageGear supported format, its ID, versions, encoding type, multi-page and alpha channel support, supported compressions, color spaces and bit depths for read and write, as well as information about ImageGear supported features and filter control parameters.

The section ImageGear Supported Digital Camera RAW Formats provides you with information about the level of ImageGear support for DC RAW formats.

If you are going to use the ImageGear alpha channels support, please review the section ImageGear Alpha Channel Support.

Encoding vs. **Compressing**

These two words are often used interchangeably in discussions of graphics file formats. Encoding is actually a broad term under which compression falls. For the sake of clarity, use these terms separately with the following intended meanings:

- **Encoding -** The manner that data is stored when uncompressed (binary, ASCII, etc.), how it is packed (e.g., 4-bit pixels may be packed at a rate of two pixels per byte), and the unique set of symbols used to represent the range of data items.
- **Compressing -** A “physical” rewriting of the graphics data so that it is represented by a smaller set of data.

Format Suitability at a Glance

The following table contains the formats that AccuSoft considers most practical for the listed type of data, and can be used as a starting point. You can use a format with the appropriate character depth and one that provides either the most efficient use of space, or the fastest loading and saving capabilities.

Bear in mind that the following table represents recommendations only. To make an informed choice, read about each format in more detail in the section ImageGear Supported File Formats Reference.

Image type	Recommended
Colors	JPEG, JPEG2000, TIFF, PNG, FlashPix
Gray-scale/many shades	TGA, TIFF, JPEG, DICOM
Monochrome / high resolution	TIFF (CCITT Group 3 compression), TIFF (CCITT Group 4 compression)
Banking data	IBM IOCA, IBM MO:DCA (ABIC compression)
Noisy	Color: PNG, TIFF, JPEG; Bi-tonal: Group 3, Group 4
Lossless	GIF, TIFF, PNG, Group 3, Group 4, JPEG (Lossless JPEG compression)
Lossy	TIFF (JPEG compression), JPEG
Extra data to store	TIFF, EXIF-JPEG, EXIF-TIFF
Iconic images	ICO
Facsimiles	Group 3, Group 4
Multimedia	PNG, GIF, JPEG, AVI, MULTIMEDIA, QuickTime
Photographic images	Adobe PSD, PCD, JPEG, FlashPix
Internet images	GIF, JPEG, PNG, JPEG2000, WBMP

Image type	Recommended
Macintosh paint programs	MAC, MAC PICT
PC paint programs and/or graphics arts	PCX, GEM, WMF, TIFF, EPS, MAC PICT, BMP
X Windows	XBM, XPM, XWD
Medical data	DICOM
CAD/Vector	EPS, DXF, CGM, DGN, HPGL, SVG
Document processing	TIFF, Adobe PDF, Adobe PS, EPS, PTOCA, TXT (ASCII Text)

ImageGear Support for Graphics File Formats

Every application that deals with images has specific kinds of data to store and interchange, from icons to photographs. The various hardware devices used to record and store the graphics data, and the hardware intended to display or print the data also affect the design of the format. This leads to a diversity of file formats. To add to this diversity, different groups of people have different ideas about how to structure and access an image, and what kind of additional information should be stored with an image. Even formats designed to store the same kind of data can differ. Other factors that affect the outcome of the design include memory considerations, storage size, accuracy, and portability.

National and international standardization organizations, such as the American National Standards Institute (ANSI) and the International Standards Organization (ISO), seek to create standards of storage for graphics data. One example is the Joint Photographic Experts Group's (JPEG) creation of the JPEG file format. Some of its intended goals were good image quality, user-chosen compression ratio, and cross-platform flexibility. When an image is called a JPEG, it is assumed to precisely follow the standardized JPEG format.

Formats known as "de facto standards" are those that begin as proprietary formats, but by the forces of the market and sometimes by good quality, become widely supported. Some examples of de facto standards are BMP, GIF, and PCD.

A third group of file formats falls somewhere between officially recognized standards and the strictly proprietary formats. These formats are created by groups of individuals with a common interest who come together to form a more unofficial standards organization. These formats are usually intended to provide an end-all industry standard so that data with the same or similar origins can be shared across different applications or platforms. One example is the TIFF format. TIFF was designed by eight computer technology companies (headed by Aldus Corporation) with the common goal of providing a standard format for storing scanned images.

ImageGear supports graphics file formats from all of these genres, providing you with a complete range of imaging capabilities, including the capture and processing of scanned images.

In addition, ImageGear functionality enable you to exchange data easily from one format to another, and to make improvements in images using powerful image-processing API.

ImageGear list of supported formats includes all of the popular formats, including recognized standards that best utilize the latest imaging technologies.

See Also:

ImageGear Supported Compressions Reference

ImageGear Supported File Formats Reference

Support for Adobe PDF/PS Formats

ImageGear provides comprehensive support for Adobe PDF and Adobe PS formats using its ImageGear PDF Component.

To use support for PDF and PostScript formats as well as manipulate and transform PDF/PS images ImageGear PDF Component should be attached to Core ImageGear.

See Also:

Using ImageGear PDF Component

PDF Component API Function Reference

Support for CAD/Vector Formats

ImageGear provides support for the following CAD/Vector formats:

- CGM
- DGN
- DWG
- DXF
- HPGL (HPGL/2)
- SVG

To work with these formats the following ImageGear components should be attached to Core ImageGear respectively:

- ImageGearVector CGM Component
- ImageGearVector DGN Component
- ImageGear CAD/Vector DWG, DXF Component
- ImageGear CAD/Vector HPGL, HPGL/2 Component
- ImageGear CAD/Vector SVG Component

To manipulate and process CAD/Vector images ImageGear Vector Component should be attached to Core ImageGear.

NOTE: ImageGear v15.2 supports CAD/Vector formats for Win32 and Win64 platforms only.

See Also:

Using ImageGear CAD/Vector Components

Vector Component API Function Reference

Support for DICOM File Format

ImageGear provides comprehensive support for DICOM formats using its ImageGear Medical Component.

To use the support for DICOM format as well as manipulate, transform and process DICOM images ImageGear Medical Component should be attached to Core ImageGear.

See Also:

Using ImageGear MD Component

MD Component API Function Reference

Support for JBIG2 File Format

ImageGear provides comprehensive support for JBIG2 format using its ImageGear JBIG2 Component.

To use the support for JBIG2 format ImageGear JBIG2 Component should be attached to Core ImageGear.

See Also:

Using ImageGear JBIG2 Component

Support for JPEG2000 File Formats

ImageGear provides support for the following JPEG2000 formats:

- JPEG2000
- JPX
- Motion JPEG2000

To use the support for JPEG2000 formats as well as ImageGear JPEG2000 Component should be attached to Core ImageGear.

See Also:

Using ImageGear JPEG2000 Component

JPEG2000 Component API Function Reference

Support for Metafile Formats

Metafiles contain vector (or geometric) specifications and bitmap pixel data. Vectors, in the realm of computer graphics, define shapes and location of shapes in terms of their relative location within the page.

ImageGear supports the following formats that contain vector data:

- WMF
- WPG
- EPS
- MAC PICT

For all formats, except WMF, the vector data is ignored; only the bitmap data is read and/or written.

NOTE: When ImageGear loads a Windows Metafile (WMF), it automatically converts each vector specification to bitmap data. The shapes and lines declared by the vector specifications appear in the image, but the original vector instructions are not saved.

Support for Multimedia File Formats

ImageGear provides support for the following multimedia formats:

- AVI
- MULTIMEDIA
- QuickTime

To use the support for multimedia formats as well as ImageGear Multimedia Component should be attached to Core ImageGear.

ImageGear Multimedia Component uses DirectShow internally to read and decode files and the types of AVI and MPEG files you can read with it depend on what files DirectShow can read, i.e. it depends on what video codecs are installed on the computer. For example, if you want to read a DivX-compressed AVI file, you can do it with the ImageGear Multimedia Component, but only if you have installed the DivX codec on your PC.

On the other way if you can view the file in Windows Media Player, you should be able to read it with the ImageGear Multimedia Component.

Support for Multi-page File Formats

ImageGear provides multi-page support for the following file formats:

- Adobe PDF
- Adobe PS
- Adobe PSD
- AVI

- BMP (OS/2 BMP only)
- CGM
- CUR
- DCX
- DGN
- DICOM
- DWG
- DXF
- GIF
- HPGL
- IBM AFP
- IBM IOCA
- IBM MO:DCA
- ICO
- IFF
- JBIG2
- MULTIMEDIA
- PCD
- PTOCA
- SVG
- TIFF
- TXT (ASCII Text)

Please see the section Loading/Saving a Document (Multi-page File) and Updating a Document (Multi-page File) to learn how to use ImageGear multi-page functionality.

To create multi-page files, simply save to an existing Adobe PDF, Adobe PS, DCX, DICOM, GIF, IBM AFP, JBIG2 or TIFF file. If the file exists, the new page is appended to the file.

NOTE: It is important to note that ImageGear treats the first page of a multi-page file as page number 1 (not 0).

ImageGear Supported Bit Depths

You may want to convert a file from one format to another. You can do this by saving the original file to its desired format by setting the `nFormatType` parameter to the appropriate value in the saving function. For more information on converting issues, see the section Loading, Saving and Acquiring Images.

When saving a file to another format, remember to ensure that the desired file format is supported for that image. For example, you cannot convert an 8-bit DCX file to an 8-bit CAL file, because the CAL format does not support 8-bit files.

In the tables on the following pages, the columns underneath the numbers indicated size in bits that can actually be saved (written). The columns underneath the letters “R” and “W” indicate that the ImageGear reads and writes, respectively, the corresponding file format.

Format	R	W	1	4	8	9-16 G	24	32	36	48	64
Adobe DNG	+					+				+	
Adobe PSD[a]	+	+	+	+	+	+	+	+		+	+
AFX	+						+				
AVI	+		+	+	+		+				
BMP	+	+	+	+	+	+	+	+[b]			
BTR	+	+	+								
CAL	+	+	+								
CGM	+	+									
CLP	+	+	+	+	+		+				
CUR[c]	+	+	+	+	+			+			
CUT	+	+			+						
DCRAW	+					+				+	
DCX	+	+	+	+	+		+				
DGN	+										
DIB	+	+	+	+	+	+[d]	+	+[e]			
DICOM	+	+	+		+	+	+	+		+	
DWG	+	+									
DXF	+	+									

Format	R	W	1	4	8	9-16 G	24	32	36	48	64
EXIF-JPEG	+	+			+	+	+	+	+		
EXIF-TIFF	+	+	+	+	+	+	+	+	+	+	+
FlashPix	+	+			+	+	+				
GEM	+		+	+	+						
GIF	+	+	+	+	+						
Group 3	+	+	+								
Group 3 2D	+	+	+								
Group 4	+	+	+								
HPGL	+										
IBM AFP	+		+	+	+		+				
IBM IOCA	+	+[f]	+	+	+		+				
IBM MO:DCA	+	+[g]	+	+	+		+				
ICO	+	+	+	+	+			+			
IFF	+	+	+	+	+		+				
IMG	+		+		+						
IMR	+		+								
IMT	+	+	+								
JBIG2	+	+	+								
JPEG	+	+			+[h]	+	+	+	+	+	
KFX	+		+								
LV	+		+								
MAC	+		+								
MAC PICT	+	+	+	+	+		+				
Motion JPEG2000	+		+	+	+	+	+	+			
MSP	+		+								

Format	R	W	1	4	8	9-16 G	24	32	36	48	64
MULTIMEDIA	+						+				
NCR	+	+	+	+[i]							
PBM	+	+	+		+	+	+			+	
PCD	+						+				
PCX	+	+	+	+	+		+				
PGM[j]	+	+			+	+					
PNG[k]	+	+	+	+	+	+	+	+		+	+
PNM[l]	+	+	+		+	+	+			+	
PPM	+	+					+			+	
QuickTime	+						+				
RAS	+	+	+	+[m]	+		+				
RAW	+		+	+	+	+	+	+			
Scitex CT[n]	+	+						+			
SGI	+	+			+	+	+	+			
SVG	+	+									
TGA	+	+	+		+	+	+	+			
TIFF[o]	+	+	+	+	+	+	+	+	+	+	+
TXT (ASCII Text)[p]	+										
WBMP	+	+	+								
WMF	+	+	+	+	+		+				
WPG	+		+	+	+						
XBM	+	+	+								
XPM	+	+	+	+	+		+[q]				
XWD	+	+	+	+	+		+				

a.48- and 64-bit images support is Read only.

b.Read only support.

c.Supports Extra ((1, 3, 4, 8)*2)-bit images also.

d.Read only support.

e.Read only support.

f.Write support for 1-bit images only.

g.Write support for 1-bit images only.

h.8-bit gray scale.

i.Read only.

j.Supports 16-bit grayscale images also.

k.Supports 48(RGB)- and 64(RGB+alpha)-bit images.

l.Supports 16-bit grayscale and 48-bit color images also.

m.Read only support.

n.The native format is CMYK. With full CMYK support enabled, you can use 32-bit images.

o.Read only supports also 3- and 6 bpp for RGB and LAB color spaces for Deflate, LZW (Lempel-Ziv-Welch), Packbits compressions and uncompressed.

p.Converts to raster image when loaded.

q.24-bit has Read only support

ImageGear Supported Compressions Reference

All imaging files compressions can be divided on three basic types:

- One-dimensional compression - the raster data is treated as one continuous data stream. Each byte read is compared to the previous byte. This compression method is not concerned with delineating lines of data.
- 2D compression can be thought of as “Differencing Compression”, where the data stored is a representation of the differences in data values from previous data values. In 2-D compression, the encoding of one line is determined by the contents of the previous line. This method of compression is best used for black-and-white images where the black pixels tend to fall into groups.
- 3D compression is a new branch of data compression aimed at the 3D models and other geometric datasets used in computer graphics, virtual reality, video games, CAD/CAM, and many scientific, engineering, and medical applications.

Existing 3D compression algorithms use both techniques adapted from the 1D and 2D cases (like wavelets, entropy coding, and predictive coding), and completely different approaches that take advantage of the properties of 3D surfaces (like Edgebreaker, Subdivision Surfaces, and triangle strips).

ImageGear supports the following compressions:

- ABIC
- ABIC_BW

- ABIC_Gray
- ASCII
- CCITT Group 3
- CCITT Group 3 2D
- CCITT Group 4
- Deflate
- FPX NOCHANGE
- FPX SINCOLOR
- Huffman
- IBM MMR
- JBIG2
- JPEG
- JPEG2000
- Lossless JPEG
- LZW (Lempel-Ziv-Welch)
- Packbits
- Progressive JPEG
- RAW
- RLE

ABIC

Full Name	IBM ABIC compression
Compression ID	IG_COMPRESSION_ABIC =16
IG Component	ABIC
Bit Depth	1
File Formats	IBM IOCA, IBM MO:DCA
IG Platforms Support	WIN32

Comments:

This is a proprietary compression scheme (owned by IBM) used on IOCA/MO:DCA. It requires the ImageGear ABIC Component to be attached.

ABIC_BW

Full Name	IBM ABIC compression
Compression ID	IG_COMPRESSION_ABIC_BW = 9
IG Component	ABIC
Bit Depth	1
File Formats	IBM IOCA, IBM MO:DCA, RAW
IG Platforms Support	WIN32

Comments:

This is a proprietary compression scheme (owned by IBM) used on IOCA/MO:DCA. Used for ABIC black and white (B/W) images. It requires ImageGear ABIC Component to be attached.

ABIC_Gray

Full Name	IBM ABIC compression
Compression ID	IG_COMPRESSION_ABIC_GRAY = 10
IG Component	ABIC
Bit Depth	1
File Formats	IBM IOCA, IBM MO:DCA, RAW
IG Platforms Support	WIN32

Comments:

This is a proprietary compression scheme (owned by IBM) used on IOCA/MO:DCA. Used for ABIC grayscale images. It requires ImageGear ABIC Component to be attached.

ASCII

Full Name	American Standard Code for Information Interchange (ASCII)

Compression ID	IG_COMPRESSION_ASCII = 23
IG Component	Core
Bit Depth	Gray level: 1, 8, 16 bpp; RGB 24, 48 bpp
File Formats	PBM, PGM, PNM, PPM
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Acronym for the American Standard Code for Information Interchange. ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Text files stored in ASCII format are sometimes called ASCII files. Text editors and word processors are usually capable of storing data in ASCII format, although ASCII format is not always the default storage format. Most data files, particularly if they contain numeric data, are not stored in ASCII format. Executable programs are never stored in ASCII format.

CCITT Group 3

Full Name	CCITT Group 3
Compression ID	IG_COMPRESSION_CCITT_G3 = 3
IG Component	Core
Bit Depth	1
File Formats	BTR, EPS, Group 3, IBM IOCA, IBM MO:DCA, IMT, LV, NCR, Adobe PDF, Adobe PS, TIFF, RAW
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

This is a 1-D version of the CCITT Group 3 compression scheme. It uses a static table of values to assign codes to run lengths. Frequently occurring run lengths are given smaller codes. (The most frequent are usually black runs of 2 or 4 pixels).

CCITT Group 3 2D

Full Name	CCITT Group 3 2D

Compression ID	IG_COMPRESSION_CCITT_G32D = 5
IG Component	Core
Bit Depth	1
File Formats	BTR, EPS, Group 3 2D, IBM IOCA, IBM MO:DCA, Adobe PDF, Adobe PS, TIFF, RAW
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

This is a 2-D version of the CCITT Group 3 compression scheme. It includes an error recovery algorithm for error transmissions; an error in one line does not translate to garbage output for the rest of the file. All modern fax machines support this format.

CCITT Group 4

Full Name	CCITT Group 4
Compression ID	IG_COMPRESSION_CCITT_G4 = 4
IG Component	Core
Bit Depth	1
File Formats	CAL, EPS, Group 4, IBM IOCA, IBM MO:DCA, IMR, IMT, KFX, LV, NCR, Adobe PDF, Adobe PS, TIFF, RAW
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

The G4 compression is two-dimensional by default. It is very similar to the G3 2D compression, but it can produce compressed images that are half the size of a G3-compressed file.

It is slower, however, and does not have the same error recovery built in that the G3 format has. The decrease in speed occurs because G4 was designed specifically for encoding disk data. For this reason, it may be advisable to use the G3-Fax compression scheme if final compression size is not crucial, but speed is.

Deflate

Full Name	Deflate compression
Compression ID	IG_COMPRESSION_DEFLATE = 14

IG Component	Core
Bit Depth	1, 2, 4, 8, 16 bpc
File Formats	Adobe PDF, PNG, Adobe PS, TIFF
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

"Zip-in-TIFF" compression. Deflate compression, sometime known as "zip" compression, uses another variant of the LZW compression method and so gives similar results, but is not restricted by any licenses.

FPX NOCHANGE

Full Name	Unchanged compression
Compression ID	IG_COMPRESSION_FPX_NOCHANGE = 13
IG Component	FlashPix
Bit Depth	Gray level: 8 bpp; RGB: 24 bpp
File Formats	FlashPix
IG Platforms Support	WIN32

Comments:

This is not a compression scheme but rather a saving mode, allowing to save FlashPix images with the same compression as they were loaded. Use this mode to change FlashPix image metadata without changing its pixels.

This compression requires the ImageGear FlashPix Component to be attached.

FPX SINCOLOR

Full Name	Single color compression
Compression ID	IG_COMPRESSION_FPX_SINCOLOR = 12
IG Component	FlashPix
Bit Depth	Gray level: 8 bpp, RGB: 24 bpp
File Formats	FlashPix
IG Platforms Support	WIN32

Comments:

This is compression is used with FlashPix images. FlashPix images are divided into tiles; when all pixels of a tile have the same color, only this single color value is saved. If a tile has pixels of different colors, the tile is saved uncompressed.

This compression requires the ImageGear FlashPix Component to be attached.

Huffman

Full Name	Huffman encoding
Compression ID	IG_COMPRESSION_HUFFMAN = 2
IG Component	Core
Bit Depth	1, 24
File Formats	NCR, PCD, TIFF
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Developed in 1952 by David Huffman, this is one of the older compression methods. The encoding and decoding processes are complex relative to today's standards, but the compression ratio can be high if the image contains many repeat data values. It is best used for images with little or no pixel noise, e.g. cartoons or drawings with large areas of the same color and intensity (like a monotone sky). This compression scheme is often used by other compression algorithms for extra compression.

The Huffman method uses a conversion table to assign codes for each value, based on frequency of occurrence. The file is scanned for all of its values, with the values and their frequency of occurrence tallied. Using a binary tree, values are paired off by frequency of occurrence, beginning with the least frequent values. As the tree progresses upward, the least occurring values at the bottom of the tree continue to be incremented a bit at a time, with one bit added for each new branch added to the tree. In the end, the values that occur the most (at the top of the tree) have the shortest codes.

A potential problem with this compression method is decoding; the file's variable-length codes can cause the dropping or adding of a bit to the end of a line, thereby throwing off subsequent lines of data.

IBM MMR

Full Name	IBM Modified Modified Read
Compression ID	IG_COMPRESSION_IBM_MMR = 15

IG Component	Core
Bit Depth	1
File Formats	IBM IOCA, IBM MO:DCA
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Compression for black and white documents, similar to CCITT Group 4.

JBIG2

Full Name	JBIG2 compression
Compression ID	IG_COMPRESSION_JBIG2 = 19
IG Component	JBIG2
Bit Depth	1
File Formats	JBIG2, Adobe PDF
IG Platforms Support	WIN32

Comments:

The JBIG2 is the most effective compression method for bitonal (B\W) images. It requires ImageGear JBIG2 Component to be attached.

JPEG

Full Name	JPEG compression
Compression ID	IG_COMPRESSION_JPEG = 6
IG Component	Core
Bit Depth	Gray level: 8, 16 bpp; RGB: 24, 36 bpp; CMYK: 32 bpp; RGB+Alpha: 32 bpp
File Formats	AFX, AVI, JPEG, DICOM, Adobe DNG, EPS, EXIF-JPEG, FlashPix, MAC PICT, Adobe PDF, Adobe PS, TIFF
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

This file compression method obtains a high compression ratio when used with detailed photographic images (its intended use). It is not a good compression choice for images with a small number of colors and high contrast edges, or for color-mapped data. Part of JPEGs success in high compression is due to the fact that it is a “lossy” compression method, meaning that the compression results in the loss of some data that is determined to be unimportant or unnecessary. This does not necessarily result in a visible reduction of image quality.

JPEG is highly flexible - it allows you to make a “quality” setting that determines the amount of loss that occurs and affects the size of the resulting compressed file.

The JPEG algorithm takes into account that the human eye is more sensitive to changes in brightness than to number of colors. Rather than saving the color data from each pixel in an image, it saves information on the rate of change of color, or “frequency information.” More loss is allowed in the color data than in the brightness data. Some of the compression of the color is achieved by converting the RGB values to YCbCr color scheme. ImageGear supports two other JPEG compression modes--Lossless JPEG and Progressive JPEG.

See Also:

Lossless JPEG, Progressive JPEG

JPEG2000

Full Name	JPEG 2000 wavelet compression
Compression ID	IG_COMPRESSION_JPEG2K = 25
IG Component	JPEG2K
Bit Depth	1..16 bpc, various colorspaces (depends on file format)
File Formats	JPEG2000, JPX, Motion JPEG2000, DICOM, Adobe PDF, Adobe PS
IG Platforms Support	WIN32, WIN64, .NET, .NET64

Comments:

This compression requires the ImageGear JPEG2000 component to be attached. See Using ImageGear JPEG2000 Component.

The JPEG2000 compression technique is based on two main processes: Tiling and Discrete Wavelet Transform (DWT).

Tiling

The term **‘tiling’** refers to the partition of the original (source) image into rectangular non-overlapping blocks (tiles), which are compressed independently as though they were entirely distinct images. All operations, including component mixing, wavelet transform, quantization and entropy coding are performed

independently on the image tiles. Tiling reduces memory requirements, and since they are also reconstructed independently, they can be used for decoding specific parts of the image rather than the entire image.

All tiles have exactly the same dimensions, except for those at the right and lower boundary of the image. Arbitrary tile sizes are allowed, up to and including the entire image (i.e., the whole image is regarded as one tile). Components with different sub-sampling factors are tiled with respect to a high-resolution grid, which ensures spatial consistency on the resulting tile components. Smaller tiles create more tiling artifacts compared to larger tiles.

Discrete Wavelet Transform (DWT)

The Discrete Wavelet Transform (DWT) is very similar to the Discrete Fourier Transform (DFT), but does not use the sine and cosine functions for analyzing the image data. The basic functions are called scaling functions and wavelets. These functions combine the fundamental feature of orthogonality (allowing the transformation and the identical reconstruction) and compact representation. This allows the analysis of image data without the windowing effects, which result from the need to handle the infinite sine function on finite image data using the DFT.

Images can be seen as two dimensional signals. They can be transformed using a two dimensional transformation. Two dimensional transformations can be achieved by separately applying one dimensional transformations to the lines and columns of an image. The DWT method splits the image into four new images in the wavelet domain; one image contains the low-pass information and the others contain high-pass information of horizontal, vertical and diagonal orientation. The transformation then applies itself recursively to the low-pass image. This can be continued until the last low-pass image contains only a single pixel.

The tile components of JPEG2000 images are decomposed into different *decomposition levels* using Discrete Wavelet Transform. These decomposition levels contain a number of subbands, which consist of *coefficients* that describe the horizontal and vertical spatial frequency characteristics of the original tile component. Power of two decompositions are allowed in the form of dyadic decomposition (in Part I).

To perform the forward DWT, the standard uses a one dimensional subband decomposition of a one dimensional set of samples into low-pass samples and high-pass samples. Low-pass samples represent a downsampled low-resolution version of the original set. High-pass samples represent a downsampled residual version of the original set, needed for the perfect reconstruction of the original set from the low-pass set.

The DWT can be *irreversible* or *reversible*. The default irreversible transform is implemented by means of the Daubechies *9-tap/7-tap filter*. The default reversible transformation is implemented by means of the *5-tap/3-tap filter*.

The standard supports two filtering modes: *a convolution-based mode* and *a lifting-based* mode. For both modes to be implemented, the signal should first be extended periodically. This periodic symmetric extension is used to ensure that for the filtering operations that take place at both boundaries of the signal, one signal sample exists and spatially corresponds to each coefficient of the filter mask. The number of additional samples required at the boundaries of the signal is, therefore, filter-length dependent.

Convolution-based filtering performs a series of dot products between the two filter masks and the extended one dimensional signal. Lifting-based filtering consists of a sequence of very simple filtering operations for which alternately odd

sample values of the signal are updated with a weighted sum of even sample values. Even sample values are updated with a weighted sum of odd sample values. For the reverse (lossless) case, the results are rounded to integer values.

The 5/3 filter allows repetitive encoding and decoding of an image without any additional loss. This is true when the decompressed image values are not clipped when they fall outside the full dynamic range (i.e. 0-255 for an 8-bpp image).

Lossless JPEG

Full Name	Lossless JPEG compression
Compression ID	IG_COMPRESSION_JPEG = 6
IG Component	Core
Bit Depth	Gray level: 8, 16 bpp; RGB: 24, 48 bpp
File Formats	DICOM, JPEG, TIFF
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Lossless JPEG is an extension to the normal JPEG standard. One of the main algorithmic differences between the two is that the lossless JPEG does not apply a Discrete Cosine Transform. Rather, it uses a Predictive scheme. For each pixel, the values of one or several neighboring pixels are added to the value of the original pixel and then subtracted from the value of the original pixel. This method yields smaller values that require fewer bits per pixel to store.

ImageGear allows you to set the number of neighboring pixels to use in calculating the “predictor value”.

LZW (Lempel-Ziv-Welch)

Full Name	Lempel-Zif-Welch (LZW) compression
Compression ID	IG_COMPRESSION_LZW = 8
IG Component	GIF/TIFF-LZW
Bit Depth	1, 4, 8, 16 bpc
File Formats	GIF, Adobe PDF, Adobe PS, TIFF
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

LZW compression works by finding patterns of data and assigning codes. It works best on highly-patterned images. Images with irregular patterning, or “noise,” are not good candidates for this type of compression.

This compression scheme is “dictionary-based”. This refers to the array of codes that identify each data pattern found in the image. The “dictionary” begins with a table that contains a code for each possible value in the image. If LZW compression is used on 8-bit images, a LZW “dictionary” is initialized with codes for 256 (28) values. As the file data is read, new values are added to the table for each unique pattern of data found. In the interest of saving space, the dictionary is not saved with the compressed file. The same dictionary is actually rebuilt when the data is decoded.

Packbits

Full Name	Packed bits compression
Compression ID	IG_COMPRESSION_PACKED_BITS = 1
IG Component	Core
Bit Depth	1, 4, 8, 16 bpc
File Formats	Adobe PSD, TIFF
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Packbits compression seeks repeated data values. Packbits is considered an RLE (run-length encoding) compression scheme because it looks for “runs” or repeated values, and tallies their number, or “length”. While its name implies that runs of bits are “packed” together, it is actually runs of bytes. It is very similar to the Macintosh Packbits compression used by Macpaint, except that the Packbits compression used for a TIFF allows the dimensions of the image to vary.

Packbits works by reducing repeated strings of the same characters into two components: the “run count” and the “run value”. The count and value are stored in one byte each. Each two-byte grouping is referred to as an RLE packet. It is not a good compression scheme for images with large color ranges, as these do not tend to have many runs of the same color.

The terms “RLE” and “Packbits” are often used synonymously.

Progressive JPEG

Full Name	Progressive JPEG compression

Compression ID	IG_COMPRESSION_PROGRESSIVE = 17
IG Component	Core
Bit Depth	Gray level: 8, 16 bpp; RGB: 24, 36 bpp; CMYK: 32 bpp; RGB+Alpha: 32 bpp
File Formats	JPEG
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Progressive JPEG is considered an extension to the JPEG standard. It produces the same kind of lossy compression as normal JPEG compression (see above), except that it saves multiple copies of the same image using different levels of quality. There is no hard limit on the number of “scans” that may be stored.

When a Progressive JPEG-compressed image is decompressed, the filter decompresses the lowest quality image first. This can be helpful for quickly displaying a version of the image that you are loading. The benefit of this compression is the fast display of an image that is recognizable. The downside is that a JPEG decompression is performed more than once.

RAW

Full Name	RAW compression (Uncompressed binary compression)
Compression ID	IG_COMPRESSION_RAW = 24
IG Component	Core
Bit Depth	Gray level: 1, 8, 16 bpp; RGB 24, 48 bpp
File Formats	PBM, PGM, PNM, PPM
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Uncompressed binary compression. PBM/PGM/PNM/PPM formats use the term “RAW“ for uncompressed binary compression, as opposed to ASCII compression.

RLE

Full Name	Run length encoding compression
Compression ID	IG_COMPRESSION_RLE = 7
IG Component	Core

Bit Depth	1, 4, 8, 16 bpc
File Formats	BMP, CLP, CUT, DCX, DICOM, GEM, IFF, MAC, MSP, PCX, Adobe PDF, Adobe PS, RAS, SGI, TGA, WMF, WPG
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

RLE (Run Length Encoding) is normally a 1-dimensional compression scheme. Working sequentially from left to right and top to bottom, it compares the value of each byte with the value of the previous byte. Each data value is recorded into a “packet” of two bytes where the first byte contains the number of times the value is repeated, and the second packet contains the actual value. The bytes in the pocket are called the “run count” and the “run value”. When an image contains many repeat values, the compression ratio is very high (for example, if every byte in a 100 byte image were the same, its size could be reduced to 2 bytes giving a 50:1 ratio). A very noisy image, or a plain ASCII text file typically does not compress well, and in fact could become larger for example, if all bytes in image are different from the ones next to them, the image doubles, because 2 bytes are used to store each byte in the image.

The terms “RLE” and “Packbits” are often used synonymously.

ImageGear Supported File Formats Reference

This section represents the reference for every imaging file format supported by ImageGear.

The following are the characteristic features for file format description:

- **Full Name** - Full name of the File Format
- **Format ID** - ImageGear constant that determines the File Format ID used for loading this file in ImageGear-based application. See the section Working with Format Filters.
- **File Extensoin(s)** - The used extensions of the File Format
- **Data Type** - The type of the File Format (raster, vector, metafile)
- **Data Encoding** - The type of data encoding. Please see the section Encoding vs. Compressing.
- **IG Multipage Support** - Shows if ImageGear supports the File Format as multipage or singlepage. Please see also the section Support for Multi-page File Formats.
- **IG Alpha Channel Support** - Shows if ImageGear supports the alpha channel for the File Format or not. Please see also the section ImageGear Alpha Channel Support.
- **IG Platforms Support** - Shows ImageGear platform versions that support the File Format.
- **ImageGear Supported versions** - Shows the versions of the File Format supported by ImageGear.
- **ImageGear Supported Features** - Shows the ImageGear Format Filter

features supported by ImageGear for this format. Please see the section Working with Format Filters.

- **ImageGear Read Support** - Provides all compressions, color spaces, channels and bit depths supported by ImageGear for the file format reading.
- **ImageGear Write Support** - Provides all compressions, color spaces, channels and bit depths supported by ImageGear for the file format writing.
- **ImageGear Filter Control Parameters** - Provides all filter control parameters supported by ImageGear for this format filter. Please see also Working with Format Filters.
- **Comments** - Some general information about format encoding and compression structure.
- **References Used** - References to the information sources for the File Format.

Currently ImageGear supports the following File Formats:

Adobe DNG	Adobe PDF	Adobe PS	Adobe PSD	AFX
AVI	BMP	BTR	CAL	CGM
CLP	CUR	CUT	DCRAW	DCX
DGN	DIB	DICOM	DWG	DXF
EPS	EXIF-JPEG	EXIF-TIFF	FlashPix	GEM
GIF	Group 3	Group 3 2D	Group 4	HD Photo
HPGL	IBM AFP	IBM IOCA	IBM MO:DCA	ICO
IFF	IMG	IMR	IMT	JBIG2
JPEG	JPEG2000	JPX	KFX	LV
MAC	MAC PICT	Motion JPEG2000	MSP	MULTIMEDIA
NCR	PBM	PCD	PCX	PGM
PNG	PNM	PPM	PTOCA	QuickTime
RAS	RAW	Scitex CT	SGI	SVG
TGA	TIFF	TXT (ASCII Text)	WBMP	WMF
WPG	XBM	XPM	XWD	

Adobe DNG

Full Name	Digital Negative file format
Format ID	IG_FORMAT_DNG = 108
File Extension(s)	*.dng
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported versions:

- Version 1.0.0.0
- Version 1.1.0.0 – fixed incompatibility in JPEG Lossless compression, added new tags

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 9..16 bpp
  - RGB: 48 bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 16 bpp
  - RGB: 48 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
RECONSTRUCT_COLORS	AT_BOOL	TRUE	TRUE, FALSE	ImageGear attempts to reconstruct full color image from the camera raw image. Otherwise, ImageGear loads raw pixel data without any processing.

Comments

This file format was developed by Adobe as a non-proprietory format for unified storage of “raw” images from digital cameras. DNG image stores unprocessed pixel data obtained from camera’s sensor, and keeps information about color, contrast and brightness adjustments, sharpening, as well as many other parameters, in its tags. This provides greater possibilities for image correction and enhancement, compared to commonly used formats such as JPEG, EXIF or TIFF.

The fact that the format is non-proprietary allows software vendors to provide support for DNG in their applications, with complete control over the image reconstruction process.

DNG extends TIFF/EP format. It adds a set of new tags for parameters that control reconstruction of full color image from the raw data.

References used

ADOBE SYSTEMS INCORPORATED. Digital Negative (DNG) specification.

Adobe PDF

Full Name	Adobe PDF (Adobe Portable Document Format)
Format ID	IG_FORMAT_PDF = 56
File Extension(s)	*.pdf
Data Type	Vector Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET

NOTE: To support PDF format ImageGear PDF Component should be attached to Core ImageGear.

ImageGear Supported versions:

- Adobe® PDF version 1.6
- Adobe® PDF version 1.5

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file
- IG_FLTR_PAGESWAPSUPPORT - page swaping in multi-page files

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp

- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_JPEG:
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JBIG2:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;

- RGB: 24 bpp;
- CMYK: 32 bpp

- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_JPEG:
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JBIG2:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale - 8 bpp

ImageGear Filter Control Parameters:

Please see the section PDF Format Filter Control Parameters Reference.

Comments:

Please see the section Using ImageGear PDF Component.

Adobe PS

Full Name	Adobe PS (Adobe PostScript)

Format ID	IG_FORMAT_PS = 102
File Extension(s)	*.ps, *.eps
Data Type	Vector Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET

NOTE: To support PS format ImageGear PDF Component should be attached to Core ImageGear.

ImageGear Supported versions:

- Adobe® PS v3

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file
- IG_FLTR_PAGESWAPSUPPORT - page swaping in multi-page files

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1,4,8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32bpp
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1,4,8 bpp;
  - Grayscale - 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32bpp
- IG_COMPRESSION_RLE:

        - Indexed RGB: 1,4,8 bpp;
        - Grayscale - 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_CCITT_G3:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_CCITT_G4:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_CCITT_G32D:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_JPEG:
        - Indexed RGB: 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_JPEG2K:
        - Indexed RGB: 1,4,8 bpp;
        - Grayscale - 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_JBIG2:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_LZW:
        - Indexed RGB: 1,4,8 bpp;
        - Grayscale - 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
    - Indexed RGB: 1,4,8 bpp;
    - Grayscale - 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32bpp
- IG_COMPRESSION_DEFLATE:
    - Indexed RGB: 1,4,8 bpp;

        - Grayscale - 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_RLE:
        - Indexed RGB: 1,4,8 bpp;
        - Grayscale - 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_CCITT_G3:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_CCITT_G4:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_CCITT_G32D:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_JPEG:
        - Indexed RGB: 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_JPEG2K:
        - Indexed RGB: 1,4,8 bpp;
        - Grayscale - 8 bpp;
        - RGB: 24 bpp;
        - CMYK: 32bpp
    - IG_COMPRESSION_JBIG2:
        - Indexed RGB: 1 bpp
    - IG_COMPRESSION_LZW:
        - Indexed RGB: 1,4,8 bpp;
        - Grayscale - 8 bpp

ImageGear Filter Control Parameters:

Please see the section PostScript Filter Control Parameters.

Comments:

Please see the section Using ImageGear PDF Component.

Adobe PSD

Full Name	PSD (Adobe Photoshop)
Format ID	IG_FORMAT_PSD = 36
File Extension(s)	*.psd
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	Supports single Alpha channel for Read only. Supports aditional Alpha channels as ImageGear Extra channels for Read only.
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 2.0
- Version 2.5
- Version 3.0
- Version 8.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;

- CMYK + Alpha: 40, 80 bpp;
- CMYK + Alpha + Extra: 8, 16 bpc;
- LAB + Alpha: 32, 64 bpp;
- Lab + Alpha + Extra: 8, 16 bpc;
- HSL + Alpha: 32, 64 bpp;
- HSL + Alpha + Extra: 8, 16 bpc;

- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha: + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Alpha: 40, 80 bpp;
  - CMYK + Alpha + Extra: 8, 16 bpc;
  - LAB + Alpha: 32, 64 bpp;
  - Lab + Alpha + Extra: 8, 16 bpc;
  - HSL + Alpha: 32, 64 bpp;
  - HSL + Alpha + Extra: 8, 16 bpc;
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Alpha: 40, 80 bpp;
  - CMYK + Alpha + Extra: 8, 16 bpc;
  - LAB + Alpha: 32, 64 bpp;
  - Lab + Alpha + Extra: 8, 16 bpc;

- HSL + Alpha: 32, 64 bpp;
- HSL + Alpha + Extra: 8, 16 bpc;

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - CMYK: 32, 64 bpp;
  - Lab: 24, 48 bpp
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - CMYK: 32, 64 bpp;
  - Lab: 24, 48 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - CMYK: 32, 64 bpp;
  - Lab: 24, 48 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
THUMBNAIL_ENABLE	BOOL	FALSE	FALSE, TRUE	When TRUE thumbnail reading function loads thumbnail image provided by PSD format.
READ_LAYER_MASK	BOOL	FALSE	FALSE, TRUE	When TRUE PSD layer mask images are loaded as usual pages of multipage file, otherwise layer masks are ignored.

Comments:

The PSD file is considered by many in the computer graphics arts community as an industry standard.

The PSD is organized into 5 major segments of data: the header, 3 informational blocks, and the bitmap data. The short header always contains a “signature” of 8PPS, as well as these fields: width, height, and bit depth of the bitmap.

The first block of informational data is called the “Color Mode Data Block”. It begins with a value for the length of the block. If the image has a palette, it is located here.

The next block is called the “Image Resources Block”. Like the previous block, it first gives the length of the block. An ID field is filled with one of many possible values that indicate the structure where the data is stored. It may contain such parameters as resolution.

The last informational block is called the “Layer and Mask Instruction Block”. After a value for the length of the block, it tells how many “layer records” follow. There is a record for each layer in the image. Each record begins with a channel ID and the length of the data in the record. After the records, a “Layer Mask” section may be stored, if applicable.

The bitmap data represents the last segment of a PSD file.

NOTE: For historic reasons, ImageGear uses the IG_COMPRESSION_PACKED_BITS constant for saving Packed Bits compressed PSD images, and uses the IG_COMPRESSION_RLE constant to report Packed Bits compression when reading PSD images. For the PSD format, the terms “Packed Bits” and “RLE” are used synonymously.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats, 2d ed. Sebastopol, CA: O’Reilly & Associates, Inc., 1996.

AFX

Full Name	Auto FX Photographic Edges
Format ID	IG_FORMAT_AFX = 49
File Extension(s)	*.afx
Data Type	Raster Image
Data Encoding	JPEG Lossy
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_JPEG - RGB: 24 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

No

Comments:

The Auto-FX file contains a small header followed by a JPEG datastream.

AVI

Full Name	MS Video for Windows video clip
Format ID	IG_FORMAT_AVI = 52

File Extension(s)	*.avi
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

NOTE: To support AVI format ImageGear Multimedia Component should be attached to Core ImageGear.

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
FILE_NAME	String	""	any string	Obsolete. Not currently used.
IMAGE_IS_KEY_FRAME	Boolean	True	True, False	The filter will set this to True or False when reading an image depending on whether or not the image is marked as a key frame in the AVI file.

Comments:

AVI files are a special case of RIFF files. RIFF is the Resource Interchange File Format. This is a general purpose format for exchanging multimedia data types that was defined by Microsoft and IBM. An AVI file ("audio/video interleave") typically contains video and optionally audio which is synchronized to the video.

ImageGear can read video frame images from uncompressed and RLE compressed AVI files using this AVI format filter. To read AVI files which use other forms of compression, it's necessary to use the MULTIMEDIA format filter instead.

BMP

Full Name	Microsoft Windows Bitmap
Format ID	IG_FORMAT_BMP = 2
File Extension(s)	*.bmp, *dib
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Only for OS/2 BMP
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64, Java

ImageGear Supported versions:

- Windows Bitmap version 5 (created for Windows 98, Windows 2000)
- Windows Bitmap version 4 (created for Windows 95, Windows NT 4.0)

- Windows Bitmap version 3 (created for Windows 3.x)
- Windows Bitmap version 2 (created for Windows 2.x)
- OS/2 Bitmap version 2
- OS/2 Bitmap version 1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 15, 16, 24 bpp;
  - RGB + Alpha: 32 bpp.
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 15, 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
`UPSIDE_DOWN`	`BOOL`	`FALSE`	`FALSE`, `TRUE`	If `TRUE` then the images will be saved upside-down
`TYPE`	`UINT`	`BMP_TYPE_BMI`	`BMP_TYPE_BMC`, `BMP_TYPE_BMI`, `BMP_TYPE_BMI2`	Type of BMP, see `BMP_TYPE_...` constants

Filter Control Parameter	Type	Default Value	Available Values	Description
COMPRESSION	DWORD	BMP_COMP_RGB	BMP_COMP_RLE4, BMP_COMP_RLE8, BMP_COMP_RGB	BMP compression, see BMP_COMP_... constants
B16_GRAY_SCANNER	BOOL	FALSE	FALSE, TRUE	Vidar 12-bit scanner options
B16_GRAY_SCANTYPE	UINT	0		Vidar 12-bit scanner options

Comments:

The BMP format for versions 2.x - 4.x contains two headers. All Windows bitmap files begin with the same first header. They proceed with a data structure containing image information (the Bitmap Information Header), and end with the actual image data. If there is a palette (1, 4, 8-bit images), it is located between the bitmap information and the bitmap image data.

The first header identifies the format as BMP, and stores the file size and the address of the image. Two additional fields, Reserved1 and Reserved2, are not used and are set to 0.

The second header, known as the “bitmap information header”, varies across the versions of Windows bitmaps. The second header for all bitmaps from version 2.x to 5.x have in common the following basic set of information: size of the secondary header in bytes, height and width of the image in pixels, the number of bit planes, the number of bits per pixel, compression scheme (0 = uncompressed, 1 = 4-bit RLE compression, 2 = 8-bit RLE compression, 3 = bitfields encoding was used), size of image in bytes, horizontal and vertical resolution in pixels per meter, the number of colors in the image, and the minimum number of important colors.

If the image is 16 or 32-bits per pixel in resolution, the compression field equals 3, and following the header are values for RedMask, GreenMask and BlueMask, rather than a palette. If the file is 4.x, there are values for an alpha component, color space type, x and y coordinates of red, green or blue endpoints, and gamma values for red, green, and blue coordinate scale values. The “ColorsImportant” field accommodates hardware that supports fewer colors than are contained by the image palette. The most significant colors are determined by counting their frequency of appearance. A value of zero means that all the colors in the image are significant.

The palette, or color table, varies in size depending on the number of colors in the image. This value is stored in the “ColorsUsed” field of the Bitmap Information Header. In the BMP format v. 3, the palette's structure is in “RGBQUAD” format. See the section entitled “Palettes” in for more information. 24-bit images do not use a palette, but rather store the color information directly in the image data.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

BTR

Full Name	Brooktrout
Format ID	IG_FORMAT_BRK = 3
File Extension(s)	*.brk, *.301,
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported versions:

Version 1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetect
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

NOTE: When saving an image, the image must have horizontal resolution of 200 DPI and vertical resolution of 100 or 200 DPI.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
COMPRESSION	WORD	BTR_COMP_G3	BTR_COMP_G3 BTR_COMP_G3_2D	Compression for saving.

Comments:

A Brooktrout file consists of a CCITT Group 3 (G3) compressed file with a 128-byte header designed by Brooktrout Technology. The header fields include a constant that identifies the file as Brooktrout, a version number, the horizontal and vertical resolutions of the image in dots/mm, the number of bits per pixel, and the number of pixels per line.

CAL

Full Name	CALS Raster
Format ID	IG_FORMAT_CAL = 4
File Extension(s)	*.cal, *.ras, *.cals
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64, Java

ImageGear Supported versions:

- Type II (Tiles made possible)
- Type I

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB : 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB : 1 bpp

ImageGear Filter Control Parameters:

No

Comments:

The CALS file format was created as a graphics format specification by The Department of Defense to standardize the data exchange of logistics support operations across the military branches and military contractors. It is mandatory for most military document-handling applications. It is no longer used solely by the military and its contractors; other government agencies and commercial businesses have also adopted this format, including the aerospace, commercial computer, and medical industries.

There are two types of CALS files: Type I, and a newer, significantly more complicated Type II. Type II supports the use of tiles. Sometimes it acts a repository for a group of Type I files. Whether or not a Type II contains Type I images determines what kind of compression scheme is used. Type I is supported by ImageGear.

Type I and Type II files begin with a header that has the same format and size. It includes information about the source and destination documents, as well as image characteristics data. The data storage units under the header are each 128 bytes in length, and are referred to as records. These are written with 7-bit ASCII characters, making it more “human-readable” than most file format headers.

The image data follows the header. In a Type II file, if the data is a series of Type I images, the images are encoded with CCITT Group 4 compression. If they are Type II files, the data may either be uncompressed or encoded with CCITT Group 4. In addition, Type II data may be stored in tiles, wherein some, all, or none of the tiles may be compressed.

The Type II file contains several more substructures than Type I. Between the header and the image data are three groups of document formatting data. Other data preceding each image (or images), are “layout information” and a “Tile Index”, that contains the address of each tile stored for the image.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

CGM

Full Name	Computer Graphics Metafile
Format ID	IG_FORMAT_CGM = 75

File Extension(s)	*.cgm
Data Type	Metafile Vector Image
Data Encoding	Character-based, Binary, Clear-text encoded
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, .NET, .NET64

NOTE: To support CGM format ImageGear CAD/Vector (CGM) Component should be attached to Core ImageGear.

ImageGear Supported versions:

- Version 1 (1987 standard)
- Version 2
- Version 3 (1992 standard)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEREADSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Write Support:

- IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
`WIDTH`	Long	512	Any positive value	Defines the DIB width of the loaded image. The DIB height is calculated using the file's scale factors.

Comments:

This is standardized platform-independent format used for the interchange of bitmap and vector data.

CGM is a very feature-rich format which attempts to support the graphic needs of many general fields (graphic arts, technical illustration, cartography, visualization, electronic publishing, and so on). While the CGM format is rich in features (many graphical primitives and attributes), it is less complex than PostScript, produces much smaller (more compact) files, and allows the interchange of very sophisticated and artistic images.

Computer Graphics Metafile was developed by experts working on committees under the auspices of the International Standards Organization (ISO) and the American Standards National Institute (ANSI). It was specifically designed as a common format for the platform-independent interchange of bitmap and vector data, and for use in conjunction with a variety of input and output devices. Although CGM incorporates extensions designed to support bitmap (called raster in the CGM specification) data storage, files in CGM format are used primarily to store vector information. CGM files typically contain either bitmap or vector data, but rarely both.

CLP

Full Name	Windows Clipboard
Format ID	IG_FORMAT_CLP = 5
File Extension(s)	*.clp
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

No

Comments:

The CLP file format represents a subset of a file format called Pictor PC Paint. The use for PC Paint format is to display images created by the PC Paint application to IBM display hardware (CGA, EGA, VGA, etc.).

The header of a CLP file is fairly short and simple, containing the file size (in bytes), and the height, width, and address of the image. The image data may be compressed or uncompressed. If it is compressed, the header contains two additional fields to give the number of bits per pixel of the packed data, and the address of the beginning of the packed run.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

CUR

Full Name	Windows cursor
Format ID	IG_FORMAT_CUR = 96
File Extension(s)	*.cur
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	Single alpha channel for read/write (see Comments).
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

Windows 3.x

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Indexed + Extra: 4+1, 8+1 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
READ_AS_RGBA	Boolean	False	True, False	If True, ImageGear reads CUR format as RGBA (RGB with alpha channel). See Comments for more detail.

Comments:

Cursor files consist of a file header (that is repeated several times), info headers, and cursor data. Cursor data contains an XOR mask bitmap and a monochrome AND mask bitmap. Whenever Windows draws a cursor, the AND bitmap is applied to whatever is on the screen. After that, the XOR bitmap is applied.

READ_AS_RGBA control parameter determines how ImageGear reads the 1-bit AND masks. If READ_AS_RGBA is `False`, ImageGear reads AND mask into “Extra“ channel. This mode preserves unchanged pixel values from the file. However, in this mode ImageGear displays only the XOR mask and ignores AND mask (Extra channel) during display, i.e. display is not transparent. If READ_AS_RGBA is TRUE, ImageGear reads CUR files as 32 bpp RGB + Alpha. This allows transparent display.

Files must be 255x255 pixels or less.

References used:

© Jörn Daub EDV-Beratung - Glashütter Weg 105 - D-22889 Tangstedt

fileformats@daubnet.com

CUT

Full Name	Dr. Halo
Format ID	IG_FORMAT_CUT = 7
File Extension(s)	*.cut, *.pal (for the separately stored palette)
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, .NET, .NET64, Unix, Unix64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_RLE - Indexed RGB: 8 bpp

ImageGear Write Support:

IG_COMPRESSION_RLE - Indexed RGB: 8 bpp

ImageGear Filter Control Parameters:

No

Comments:

The Dr. Halo file format is associated with the HALO Image File Format Library, the Dr. Halo III paint program, and other applications created by Media Cybernetics.

This format consists of two separate files, one with an extension of .CUT and the other with an extension of .PAL. The .CUT file contains the image data and the .PAL file contains the color palette.

The .CUT file begins with a simple header of just three data fields: width and height of the image data (pixels by scan lines), and a reserved field intended for use with any future expansions of the header. The image data is always RLE-encoded and follows the header.

The palette file (.PAL) contains a header, with information about the type of palette used, and the size and maximum values of the Red, Green, and Blue components. The palette can be hardware-specific, in which case it contains additional data.

When a CUT image is loaded into ImageGear, the palette is initialized to a gray-scale ramp. In order to achieve the original colors of the palette (PAL file), it must be loaded separately into the HIGEAR image.

When saving an image into the Dr. Halo format, ImageGear creates a .CUT file. In order to save the palette (PAL file), it must be saved separately.

See the section RLE for more information.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

DCRAW

Full Name	Digital camera RAW file
Format ID	IG_FORMAT_DCRAW = 104
File Extension(s)	*.*
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported versions:

Please see the section ImageGear Supported Digital Camera RAW Formats.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Gray level: 10..16 bpp;
  - RGB: 48 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
RECONSTRUCT_COLORS	AT_BOOL	TRUE	TRUE, FALSE	If TRUE, ImageGear attempts to reconstruct full color image from the camera raw image. Otherwise, ImageGear loads raw pixel data without any processing.

Comments:

Please see the section Working with Digital Camera RAW formats in Using ImageGear chapter.

DCX

Full Name	Paintbrush (Intel multi-page FAX format)
Format ID	IG_FORMAT_DCX = 8
File Extension(s)	*.dcx
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;

- RGB: 24 bpp

- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_COMPRESSED	BOOL	TRUE	FALSE, TRUE	This parameters specify either compress output image or not. TRUE value cause to RLE compress image. FALSE cause to write image uncompressed.
ADD_IMAGE	BOOL	TRUE	FALSE, TRUE	If this parameter is TRUE then image is added as additional page of multipage image. If FALSE then the new image with single page is written

Comments:

This file format was designed to allow multiple PCX files to be stored in one file. This is especially desirable for multi-page faxes (for which the PCX format is often used). Up to 1024 PCX images can be stored in one DCX file.

The DCX construct begins with a simple header, then the PCX files are simply stored end-to-end, complete with their individual headers and palettes. An array in the header called “Pagetable” contains offsets to each PCX. The one piece of vital information not stored with the PCX files under a DCX is their original filenames. See the PCX section of this manual for more about PCX files.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

DGN

Full Name	MicroStation Drawing (Drawing format used by MicroStation construction design software)
Format ID	IG_FORMAT_DGN = 73
File Extension(s)	*.dgn
Data Type	Vector image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, .NET, .NET64

NOTE: To support DGN format ImageGear CAD/Vector (DGN) Component should be attached to Core ImageGear.

ImageGear Supported versions:

- DGN_VERSION_UNKNOWN
- DGN_VERSION_DGN2
- DGN_VERSION_DGN1_2D
- DGN_VERSION_DGN1_3D
- DGN_VERSION_CEL1_2D
- DGN_VERSION_CEL1_3D

ImageGear Supported Features:

IG_FLTR_DETECTSUPPORT - autodetection

IG_FLTR_PAGEREADSUPPORT - single page file reading

IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
BACKCOLOR	Long	0	Any positive value	This parameter is used to set background color. If nBackGroundColor=0, then a black background color is set, otherwise a white background color is set.
WIDTH	Long	512	Any positive value	Defines the DIB width of the loaded image. The DIB height is calculated using the file's scale factors.
FILENAME	String	""	N/A	Obsolete. Not currently used.

Comments:

DGN is a CAD file format used by Intergraph IGDS CAD software. It is not as commonly used as some other CAD file formats, but it has an open specification.

DIB

Full Name	DIB (Microsoft Windows Device Independent Bitmap)
Format ID	IG_FORMAT_DIB = 9
File Extension(s)	*.dib
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, .NET, .NET64

ImageGear Supported versions:

See BMP format description.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 15, 16, 24 bpp;
  - RGB + Alpha: 32 bpp.
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 15, 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
UPSIDE_DOWN	BOOL	FALSE	FALSE, TRUE	If TRUE then the images will be saved upside-down
TYPE	UINT	BMP_TYPE_BMI	BMP_TYPE_BMC, BMP_TYPE_BMI, BMP_TYPE_BMI2	Type of BMP, see BMP_TYPE_... constants
COMPRESSION	DWORD	BMP_COMP_RGB	BMP_COMP_RLE4, BMP_COMP_RLE8, BMP_COMP_RGB	BMP compression, see BMP_COMP_... constants
B16_GRAY_SCANNER	BOOL	FALSE	FALSE, TRUE	Vidar 12-bit scanner options
B16_GRAY_SCANTYPE	UINT	0		Vidar 12-bit scanner options

Comments:

DIB is a synonym for the BMP format.

DICOM

Full Name	DICOM (Digital Imaging & Communication in Medicine)
Format ID	IG_FORMAT_DCM = 48
File Extension(s)	*.dicm, *.dcm
Data Type	Raster or vector image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix (Linux), .NET, .NET64

NOTE: To support DICOM format ImageGear Medical Component should be attached to Core ImageGear.

ImageGear Supported versions:

- DICOM 3.0, 1991 - 2006

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file (only deletion of last page is supported)
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 1, 8, 9-16, 32 bpp;
  - Indexed RGB: 8 bpp, 16 bpp (converted to 24-bit RGB during loading);
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossy):
  - Grayscale: 8, 9-12 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossless):

- Grayscale: 8, 9-16 bpp;
- RGB: 24 bpp

- IG_COMPRESSION_JPEG2K:
  - Grayscale: 8, 9-16 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8, 9-16 bpp;
  - Indexed RGB: 8 bpp, 16 bpp (converted to 24-bit RGB during loading);
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8, 9-16, 32 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossy):
  - Grayscale: 8, 9-12 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 9-16 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG2K - 8, 9-16, 24, 48 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8, 9-16 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp

NOTE: To use JPEG2000 compression scheme you should attach ImageGear JPEG2000 Component.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
DETECT_CONTINUOUS_RLE	BOOL	TRUE	TRUE, FALSE	Set to TRUE for ImageGear to detect and load images where RLE runs across row boundaries.*)
LOAD_SAVE_PIXDATA_TAG	BOOL	FALSE	TRUE, FALSE	Set to TRUE to allow loading/saving pixel data to/from DataSet rather than to/from ImageGear DIB. **)
LOAD_SYNTAX	INT	MED_DCM_TS_AUTODETECT	MED_DCM_TS_ constants. See file DataDict.h	Transfer Syntax to load DCM image with.
LOAD_MASKALPHACHANNEL	BOOL	TRUE	TRUE, FALSE	Rasters are masked off
LOAD_PAGENUMBER	UINT	1		Page number to load
LOAD_CONVERTTO8G	BOOL	FALSE	TRUE, FALSE	Convert 9-16 bit gray to 8 on load
LOAD_DETECTSKIPDIMSE	BOOL	FALSE	TRUE, FALSE	Set to TRUE if files may contain DIMSE commands - that could throw off the format detection
LOAD_USE_8x8_LUT	BOOL	TRUE	TRUE, FALSE	Set to TRUE to use 8x8 display LUT. Otherwise, use image's palette (the mechanism that was used in IG v15.0 and earlier). ***)

Filter Control Parameter	Type	Default Value	Available Values	Description
LOAD_CONCAT_REPEATED_DE	BOOL	FALSE	TRUE, FALSE	TRUE to concatenate repeated data elements into one data element during loading. ****)
LOAD_USE_AUTO_WL_FOR_8G	BOOL	TRUE	TRUE, FALSE	Set to TRUE to use auto window/level for 8g images, if VOI LUT is not present. *****)
SAVE_SYNTAX	INT	MED_DCM_TS_DEFAULT	MED_DCM_TS_ constants. See file DataDict.h	Transfer Syntax to save DCM image with.
SAVE_GROUPLENGTHS	UINT	TRUE	TRUE, FALSE	TRUE=include FALSE=exclude
SAVE_ASPART10	BOOL	TRUE	TRUE, FALSE	TRUE will cause the Header to be saved
SAVE_PLANARCONFIG	INT	MED_DCM_PLANAR_PIXEL_BY_PIXEL	MED_DCM_PLANAR_... constants	Planar configuration for saving image: MED_DCM_PLANAR_...
SAVE_SMALLEST	BOOL	FALSE	TRUE, FALSE	Include/exclude Smallest Pixel Value when saving image.
SAVE_LARGEST	BOOL	FALSE	TRUE, FALSE	Include/exclude Largest Pixel Value when saving image.
SAVE_JPGQUALITY	UINT	70	1 - 100	JPEG Quality setting 1-100

*) This parameter specifies what to do with RLE compressed images where RLE runs across row boundaries. If it is set to TRUE, ImageGear tries to detect and load images where RLE runs across row boundaries. Otherwise, ImageGear truncates any row overruns and decodes each row separately. This parameter does not affect the loading of properly encoded images, where each row is encoded separately.

**) When LOAD_SAVE_PIXDATA_TAG is TRUE, ImageGear does not read pixel data into a DIB, but rather creates an empty DIB, so the image cannot be displayed. When reading compressed image, IG Medical treats PixelData tag as a Sequence, and places actual binary data into Item tags. This corresponds to the structure of compressed PixelData in DICOM files. When using this parameter for writing, make sure that Transfer Syntax matches actual Transfer Syntax of the PixelData element.

***) Parameter LOAD_USE_8x8_LUT specifies the mechanism for display contrast adjustments of 8-bit grayscale images. Set to TRUE (default) to use 8x8 display LUT. Otherwise, use image's palette (the mechanism that was used in IG v15.0 and earlier).

****) Parameter LOAD_CONCAT_REPEATED_DE allows to load incompliant DICOM images where some data elements are cut into several repeated data elements. Instead of one data element containing an array of values, the data set contains several data elements, with the same group/element numbers pair, containing portions of the array. Specifically, there are images with look up tables and palettes stored in this way. Set LOAD_CONCAT_REPEATED_DE parameter to TRUE to concatenate repeated data elements into one data element during loading. Otherwise (default), ImageGear loads all of the repeated elements in the same way as they are located in the file. Note that ImageGear does not allow saving of repeated data elements. If a data set contains repeated data elements, ImageGear will only write first repeated element to the file.

*****) Parameter LOAD_USE_AUTO_WL_FOR_8G affects loading of 8-bit grayscale images that do not have a VOI LUT (either a LUT sequence or window center/width values). Set to TRUE to use auto window/level for these images. Set to FALSE to apply no window/levelling (set contrast range to 0...255).

Comments:

DICOM is a public standard created to provide a flexible and expandable means for storing, sharing, and transporting digital medical images. Today DICOM is the standard for medical imaging throughout the world.

DICOM image (alternately called Data Set) contains an ordered collection of attributes referred to as “Data Elements” that are related to one or more images. Each Data Element (DE) describes a single attribute of the image, patient, or study. The images themselves are also stored in DEs.

Each DICOM Data Set is transported through the Network, and consequently, stored in a file, using one of the defined Transfer Syntaxes. The Transfer Syntax of the DICOM file indicates whether the file uses Big Endian or Little Endian byte order, whether the image data is compressed or uncompressed, and if the DICOM Data Set uses Explicit or Implicit Value Representation (VR).

The Data Element (DE) is made up of the following parts:

- Tag
- Value Representation (optional)
- Value Lenth
- Value

Tag field identifies the type of information that is contained in the Value field (where the actual data is stored). The DICOM Data Dictionary (Part 6 of the specification) defines all possible public Data Element Tags that may be used.

DICOM also allows applications to define and use private Tags and thus define their own Data Elements. Value Representation (VR) specifies the format of the Data Element Value, such as UL (unsigned long), ST (Short Text) or PN (Person Name). If Implicit Transfer Syntax is used, the VR field is omitted. It can be obtained from the standard or private Data Dictionary. Value Lenth is the lenth (in bytes) of the Data field.

DICOM also allows embedding (nesting) Data Sets within Data Sets. Nested Data Sets are implemented using a “Sequence of Items” (SQ), which is a special type of Data Element.

A special group of tags at the beginning of the file allows application to recognize the file as a DICOM image file, and includes information needed to decode the file (Transfer Syntax), and serial numbers to help locate and keep track of each field. This header is referred to as “File Meta Information Header”, or “Part 10 Header” (since it is defined in Part 10 of the DICOM standard). While the standard clearly states that all DICOM image files must include a Part 10 Header, in reality one finds that many do not. Instead, the average DICOM image file is a simple data stream capture of the data into a file. This is called a “Raw Data” DICOM image file. In order to read a Raw DICOM the Transfer Syntax must be guessed at using a Transfer Syntax detection algorithm.

The actual pixel data for a DICOM image is stored in a Data Element, just like any other DICOM information. Image parameters, such as dimensions, bit depth, photometric interpretation etc., are also stored in Data Elements. The Tag for Pixel Data is called “Pixel Data” and has the Tag Number 7FE0, 0010. The Data Field in this Data Element contains all pixels for the image. Depending on the image’s Transfer Syntax, the Pixel Data can be compressed or uncompressed.

References Used

Digital Imaging and Communication in Medicine (DICOM). Published by: National Electrical Manufacturers Association: http://medical.nema.org/dicom.html.

DWG

Full Name	DWG (AutoDesk AutoCAD)
Format ID	IG_FORMAT_DWG = 69
File Extension(s)	*.dwg
Data Type	Vector/CAD image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, .NET, .NET64

NOTE: To support DWG format ImageGear CAD/Vector (DWG, DXF) Component should be attached to Core ImageGear.

ImageGear Supported versions:

- Read: Versions 2.5 through 2007
- Write: Versions 12 through 2007

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Write Support:

IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
BACKCOLOR	Long	0	Any positive value	This parameter is used to set background color. If nBackGroundColor=0, then a black background color is set, otherwise a white background color is set.
WIDTH	Long	512	Any positive value	Defines the DIB width of the loaded image. The DIB height is calculated using the file's scale factors.
FONTPATH	String	“”	Any string up to 262 characters	Defines a path to the vector font files. Format - <Path1>;< Path2>;... .
PASSWORD	String	“”	Any string up to 262 characters	Used to set password for encrypted files before reading.

Filter Control Parameter	Type	Default Value	Available Values	Description
VERSION	String	ACAD2004	ACAD12 ACAD13 ACAD14 ACAD2000 ACAD2004 ACAD2007	Used to specify version to save the file as.
SHOWHIDDEN	Boolean	False	True, False	If true, hidden layers are shown by default.
TRUETYPE	Boolean	False	True, False	If true, TrueType font (Arial) are used for text drawing, otherwise vector fonts are used.
TTFWIDTH	Double	0	Any positive value	Relative default TrueType font width.
LINEWIDTHDEF	Double	0	Any positive value	Default line width.
FILENAME	String	""	N/A	Obsolete. Not currently used.

Comments

DWG, short for "DraWinG", is a CAD file format which serves as the native file format for drawings created in AutoCAD.

DXF

Full Name	DXF (AutoDesk AutoCAD)
Format ID	IG_FORMAT_DXF = 70
File Extension(s)	*.dxf
Data Type	Vector/CAD image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, .NET, .NET64

NOTE: To support DXF format ImageGear CAD/Vector (DWG, DXF) Component should be attached to Core ImageGear.

ImageGear Supported versions:

- Read: Versions 2.5 through 2007
- Write: Versions 12 through 2007

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Write Support:

- IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
BACKCOLOR	Long	0	Any positive value	This parameter is used to set background color. If nBackGroundColor=0, then a black background color is set, otherwise a white background color is set.
WIDTH	Long	512	Any positive value	Defines the DIB width of the loaded image. The DIB height is calculated using the file's scale factors.
FONTPATH	String	“”	Any string up to 262 characters	Defines a path to the vector font files. Format - <Path1>;< Path2>;... .
PASSWORD	String	“”	Any string up to 262 characters	Used to set password for encrypted files before reading.

Filter Control Parameter	Type	Default Value	Available Values	Description
VERSION	String	ACAD2004	ACAD12 ACAD13 ACAD14 ACAD2000 ACAD2004 ACAD2007	Used to specify version to save the file as.
SHOWHIDDEN	Boolean	False	True, False	If true, hidden layers are shown by default.
TRUETYPE	Boolean	False	True, False	If true, TrueType font (Arial) are used for text drawing, otherwise vector fonts are used.
TTFWIDTH	Double	0	Any positive value	Relative default TrueType font width.
LINEWIDTHDEF	Double	0	Any positive value	Default line width.
FILENAME	String	""	N/A	Obsolete. Not currently used.

Comments

DXF is an AutoDesk graphics format (Drawing eXchange Format). Most CAD programs (PC-based) support this format. Compared to DWG, it is simpler and more broadly supported among different software.

EPS

Full Name	Encapsulated PostScript File
Format ID	IG_FORMAT_EPS = 10
File Extension(s)	*.eps (may or may not contain preview image), *.epi (contains preview image), *epsf
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix (full support with ImageGear PDF component only), Unix64, MAC, .NET

ImageGear Supported versions:

- Version 3.0
- Version 2.0
- Version 1.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_PREVIEW	BOOL	FALSE	TRUE, FALSE	If TRUE then EPS preview will be saved
START	DWORD	0		Position of start.
FITTING_METHOD	UINT	IG_EPS_FIT_ACTUAL	IG_EPS_FIT_PAGE, IG_EPS_FIT_ACTUAL, IG_EPS_FIT_SET	Fitting method, see IG_EPS_FIT_... constants
PIXEL_TO_PIXEL	BOOL	FALSE		
PAGE_WIDTH	AT_DIMENSION	8500		Page width (100ths of an inch)
PAGE_HEIGHT	AT_DIMENSION	11000		Page height (100ths of an inch)
MARGINS	AT_RECT	{250, 250, 250, 250}		Margins (100ths of an inch)
X_DPI	UINT	300		X resolution
Y_DPI	UINT	300		Y resolution
TEXT_ENCODING	UINT	IG_PDF_TEXTENC_ASCII_HEX	IG_PDF_TEXTENC_NONE, IG_PDF_TEXTENC_ASCII_85, IG_PDF_TEXTENC_ASCII_HEX	Text encoding methos, see IG_PDF_TEXTENC_ constants

Comments

“PostScript” refers to a widely-supported general-purpose computer language that encodes text and graphics files for sharing with the many different hardware devices that support it. The full name for this language is “PostScript Page Description Language” (PDL).

An Encapsulated PostScript file stores (encapsulates) graphical or photographic images from a larger PostScript file. ImageGear currently supports the reading of EPS image of any bit depth as long as the preview image is TIF, uncompressed. ImageGear currently supports the writing of monochrome (1-bit) and grayscale (8-bit gray level) EPS images only.

The EPS file format begins with a PostScript language header. The data herein identifies the format as EPS, and gives the image title, creator, creation date, size and position of the image. Each line begins with a percent sign (%), which is normally interpreted in the PostScript language as the beginning of a comment line. Within the context of the EPS header, it takes on a different meaning.

Following the header is a block of PostScript code, which accomplishes the actual creation of the image.

The format proceeds with the bitmap data, or “graphics screen representation”.

The EPI version of the EPS format, the version supported by ImageGear, contains an abridged interpretation of the image that is appended to the end of the file. It is usually smaller, and contains a lower resolution. One of the benefits of a preview image is that an application does not need to be able to interpret PostScript in order to display the image. Preview images are created with one of four file format types: TIFF, WMF, MAC PICT, and EPS.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

EXIF-JPEG

Full Name	Exchangeable image file format (EXIF-JPEG)
Format ID	IG_FORMAT_EXIF_JPEG = 71
File Extension(s)	*.jpg, *.xif
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Read/write
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported versions:

- Version 1.0 (1996)
- Version 1.1 (1997)

- Version 2.0 (1998)
- Version 2.1 (1998)
- Version 2.2 (2002)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_JPEG (lossy, progressive):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - RGB + Alpha: 32 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp;
  - CMYK: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_JPEG (lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (progressive):
  - Grayscale: 8, 12 bpp;
  - RGB: 24 bpp;

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
FLASHPIX_READY	BOOL	FALSE	TRUE, FALSE	If TRUE the image will be saved with 64 interoperability, alowing lossless conversion to FlashPix format.
SAVE_JFIF_SEGMENT	BOOL	FALSE	TRUE, FALSE	If TRUE the JFIF segment will be saved with image
SAVE_THUMBNAIL	BOOL	TRUE	TRUE, FALSE	If TRUE the thumbnail will be saved with image
THUMBNAIL_WIDTH	UINT	160	Any positive value	The width of thumbnail
THUMBNAIL_HEIGHT	UINT	120	Any positive value	The height of thumbnail
THUMBNAIL_COMPRESSED	BOOL	TRUE	TRUE, FALSE	For JPEG compressed EXIF. If TRUE the thumbnail will be JPEG compressed
LOAD_SCALE_DENOM	UINT	1	1, 2, 4, 8	If this parameter is set to any other than default value, ImageGear loads reduced version of the image, width and height of which are scaled by 1/ load_scale_denom. This mode can be used for image preview, especially for those images that do not have embedded thumbnails, or where embedded thumbnails are smaller than desired. This mode allows you to make loading process 2-4 times faster.*

* LOAD_SCALE_DENOM parameter affects the following formats: JFIF-JPEG, EXIF-JPEG. It does not affect TIFF-JPEG and other formats containing JPEG stream. Only Lossy and Progressive compressed images are supported. It affects all image loading functions and all image info functions.

Comments

The EXIF file format is based on existing formats. There are two kinds of EXIF format: compressed and uncompressed. Compressed EXIF is recorded in JPEG format with EXIF header saved in APP1 and APP2 marker segments. The APP2 segment is used when recording FlashPix extensions.

Uncompressed EXIF is recorded in TIFF Rev. 6.0 formats with two pages - the first is the main image, the second is a thumbnail (if it present). The EXIF header data is stored in TIFF 6.0 format for both compressed and uncompressed EXIF and include EXIF information (that is necessary) and GPS information (that is optional). Information specific to the camera system and not defined in TIFF is stored in private tags registered for EXIF. ImageGear EXIF support allows you to retrieve this information and send it to an application level and vice versa.

The EXIF image file specification also specifies the method for recording thumbnails. The reason for using the TIFF Rev. 6.0 tag format in the compressed file APP1 segment is to facilitate exchange of attribute data between EXIF compressed and uncompressed files.

Although the standard only allows uncompressed and JPEG-compressed EXIF images, and RGB colorspace, there are EXIF images that use other compressions and colorspaces. In a sence, these images can be considered as JPEGs or TIFFs with additional EXIF metadata. ImageGear supports reading of such non-standard images, as well as writing of Exif images with some non-standard compressions and colorspaces.

References used

Digital Still Camera Image File Format Standard (Exif) Version 2.0, Nov 1997, Japan Electronic Industry Development Association.

Digital Still Camera Image File Format Standard (Exif) Version 2.1, June 1998, Japan Electronic Industry Development Association.

EXIF-TIFF

Full Name	Exchangeable image file format (EXIF-TIFF)
Format ID	IG_FORMAT_EXIF_TIFF = 74
File Extension(s)	*.tif, *.xif
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Read only
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported versions:

- Version 1.0 (1996)
- Version 1.1 (1997)
- Version 2.0 (1998)
- Version 2.1 (1998)
- Version 2.2 (2002)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;
  - RGB + Alpha + Extra: 8, 12, 16 bpc;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc.
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;
  - RGB + Alpha + Extra: 8, 12, 16 bpc;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc.
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;

        - RGB + Alpha + Extra: 8, 12, 16 bpc;
        - Lab + Extra: 8, 12, 16 bpc;
        - CMYK + Extra: 8, 12, 16 bpc.
    - IG_COMPRESSION_DEFLATE:
        - Indexed RGB: 1, 4, 8 bpp;
        - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
        - RGB: 24, 36, 48 bpp;
        - Lab: 24, 36, 48 bpp;
        - CMYK: 32, 48, 64 bpp;
        - RGB + Alpha: 8, 12, 16 bpc;
        - RGB + Alpha + Extra: 8, 12, 16 bpc;
        - Lab + Extra: 8, 12, 16 bpc;
        - CMYK + Extra: 8, 12, 16 bpc.
    - IG_COMPRESSION_HUFFMAN:
        - Indexed RGB: 1 bpp;
        - Grayscale: 1 bpp;
    - IG_COMPRESSION_CCITT_G3:
        - Indexed RGB: 1 bpp;
        - Grayscale: 1 bpp;
    - IG_COMPRESSION_CCITT_G4:
        - Indexed RGB: 1 bpp;
        - Grayscale: 1 bpp;
    - IG_COMPRESSION_CCITT_G32D:
        - Indexed RGB: 1 bpp;
        - Grayscale: 1 bpp;
    - IG_COMPRESSION_JPEG (Lossy):
        - Grayscale: 8, 12 bpp;
        - RGB: 24, 36 bpp;
    - IG_COMPRESSION_JPEG (Lossless):
        - Grayscale: 8, 16 bpp;
        - RGB: 24 bpp;

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_PACKED_BITS:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_LZW:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_DEFLATE:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_JPEG (Lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
- IG_COMPRESSION_JPEG (Lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp;

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_ THUMBNAIL	BOOL	TRUE	TRUE, FALSE	If TRUE the thumbnail will be saved with image
THUMBNAIL_ WIDTH	UINT	160	Any positive value	The width of thumbnail
THUMBNAIL_ HEIGHT	UINT	120	Any positive value	The height of thumbnail

Comments

The EXIF file format is based on existing formats. There are two kind of EXIF format: compressed and uncompressed. Compressed EXIF is recorded in JPEG format with EXIF header saved in APP1 and APP2 marker segments. The APP2 segment is used when recording FlashPix extensions.

Uncompressed EXIF is recorded in TIFF Rev. 6.0 formats with two pages - the first is the main image, the second is a thumbnail (if it present). The EXIF header data is stored in TIFF 6.0 format for both compressed and uncompressed EXIF and include EXIF information (that is necessary) and GPS information (that is optional). Information specific to the camera system and not defined in TIFF is stored in private tags registered for EXIF. ImageGear EXIF support allows you to retrieve this information and send it to an application level and vice versa.

The EXIF image file specification also specifies the method for recording thumbnails. The reason for using the TIFF Rev. 6.0 tag format in the compressed file APP1 segment is to facilitate exchange of attribute data between EXIF compressed and uncompressed files.

Although the standard only allows uncompressed and JPEG-compressed EXIF images, and RGB colorspace, there are EXIF images that use other compressions and colorspaces. In a sence, these images can be considered as JPEGs or TIFFs with additional EXIF metadata. ImageGear supports reading of such non-standard images, as well as writing of Exif images with some non-standard compressions and colorspaces.

References used

Digital Still Camera Image File Format Standard (Exif) Version 2.0, Nov 1997, Japan Electronic Industry Development Association.

Digital Still Camera Image File Format Standard (Exif) Version 2.1, June 1998, Japan Electronic Industry Development Association.

FlashPix

Full Name	FlashPix
Format ID	IG_FORMAT_FPX = 50
File Extension(s)	*.fpx
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Read only, converted to RGB during loading.
IG Platforms Support	WIN32

NOTE: To support FlashPix format ImageGear FlashPix Component should be attached to Core ImageGear.

ImageGear Supported versions:

Version 1.0.0

Version 1.0.1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp (converted to RGB during loading);
  - RGB + Alpha: 16 bpp (converted to RGB during loading);
- IG_COMPRESSION_FPX_SINCOLOR:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp (converted to RGB during loading);

- RGB + Alpha: 16 bpp (converted to RGB during loading);

- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
- IG_COMPRESSION_FPX_NOCHANGE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
- IG_COMPRESSION_FPX_SINCOLOR:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
DISABLETRANSFORMS	BOOL	FALSE	TRUE, FALSE	Set to TRUE to load the image without applying geometric and color transforms that are specified in the metadata
LOADOPTION	MODE	IG_FPX_LOAD_DEFAULT	`IG_FPX_LOAD_DEFAULT = 0, IG_FPX_LOAD_NOLESS = 1, IG_FPX_LOAD_NOGREATER = 2`	`Allows to select a resolution according to specified page width and height. *)`
PAGEBACKGROUND	RGBQUAD	{0, 0, 0, 0};	RGBQUAD	Background to use for images that have Alpha channel.

Filter Control Parameter	Type	Default Value	Available Values	Description
PAGEHEIGHT	AT_DIMENSION	0	AT_DIMENSION	See LOADOPTION
PAGEWIDTH	AT_DIMENSION	0	AT_DIMENSION	See LOADOPTION
QUALITY	UINT	70	1...100	Quality to use for saving JPEG compressed FPX images
SAVEFROMDIB	BOOL	TRUE	TRUE, FALSE	TRUE to save pixel data from HIGEAR. FALSE to save pixel data that were loaded from FlashPix image and stored internally. This allows changing FlashPix metadata, including geometric and color transforms, and saving the image without changing its original pixel data.
SETSCANNEDIMGSIZE	BOOL	FALSE	TRUE, FALSE	Use HIGEAR resolution to set ScannedImageSize tag when saving the image

*) Set to `IG_FPX_LOAD_DEFAULT` to load the highest (or the only) resolution in the resolution hierarchy. Set to `IG_FPX_LOAD_NOLESS` to load the lowest resolution which dimensions are not less than `PAGEWIDTH` and `PAGEHEIGHT` parameters. Set to `IG_FPX_LOAD_NOGREATER` to load the highest resolution which dimensions are not greater than `PAGEWIDTH` and `PAGEHEIGHT` parameters.

Comments

The FlashPix File Format was created to make the desktop handling of digital color photographic images easy, enjoyable, affordable, and commonplace. It was developed and published by Eastman Kodak Company in collaboration with Hewlett-Packard, Live Picture Inc. and Microsoft.

Kodak is marketing the Kodak Image Magic system, which includes such components as a FlashPix CD, digital cameras, scanners, and print services – that can be accessed through the internet, or through your local film developer. You do not, however, have to use digital equipment to put your photographs onto CD. You can still take your "old-fashioned" camera film to a film developer and request that it be sent to Kodak for transfer to a CD. You can also bring in existing negatives and have them transferred to CD. Ultimately, Kodak envisions a world where amateur PC users can put their latest photographs into digital storage, improve them and enhance them, drop them into card or calendar templates, and E-mail them off to Grandma and Grandpa's home computer.

When you think FlashPix file format, three major structural features should come to mind: Source Plus Transform storage scheme, hierarchy of resolutions, and tiled storage. These three key features best illustrate the purpose of the format and also set it apart from most others.

- The Source Plus Transform storage scheme refers to Kodak's use of

"transformation parameters". This set of non-image data tags (fields) stores the last set of FlashPix-specific transformations that were made to the image. In this unique storage scheme the bitmap data and the transformations for it are stored separately. This keeps the bitmap data in its original state. The bitmap data, or "source image" of a FlashPix image is never directly altered by a FlashPix transformation. One of the reasons for this structure is that it allows the end-user of a FlashPix application always to be able to return to the source image if their transformations have gotten out-of-hand. Transformation parameters can be applied (enabled) to the source image when the image is being displayed, or they can be "turned off" (disabled) so that the unaltered source image is displayed. The API functions `IG_FPX_enable()` and `IG_FPX_disable()` are used for this purpose, respectively. Additionally, you can "burn in" the transformations directly to the DIB, if so desired.

- The "hierarchy of resolutions" refers to the multiple pages with which FlashPix stores several versions of the same image - each with a different resolution. The highest resolution image represents the first image or "page" of a FlashPix file. Each subsequent image is at a lower resolution. The resolution hierarchy was developed in the interest of optimizing speed during image loading. The lower resolution images can be used for display, depending on the capability of the monitor, and the highest resolution image will always be used for printing. Kodak likes to refer to this storage as being like a pyramid.

The FlashPix is available as an ImageGear component, which brings you the core of the FlashPix features:

- a full hierarchy of resolutions for each image,
- a "Source Plus Transform" storage scheme for easy "undos" and no loss of image quality, no matter how many transformations are applied,
- a choice of JPEG or Single Color compression,
- a details-oriented stash of non-image data (title, subject, people in the picture, creation date, etc.),
- thumbnail support,
- tiled storage,
- and the use of all ImageGear functions to load, save, and transform FlashPix images.
- a special FlashPix-specific API, including functions that make reading and writing of non-image information easy,
- the ability to perform all FlashPix transformations on any ImageGear image,
- the ability to perform ImageGear transformations without losing FlashPix transformations.

GEM

Full Name	GEM Raster
Format ID	IG_FORMAT_GEM = 13
File Extension(s)	*.gem
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_RLE - 1, 4, 8 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

No

Comments

A deal with the creators of Ventura Publisher made this a significant format in the desktop publishing arena. It was also distributed by PC systems manufacturers and was the native bitmap format of the Atari ST system.

The structure of a GEM Raster image file begins with a fixed-length header and is followed by the bitmap data.

The data fields of the header include the version number (always 1), image width (in pixels), and image height (in scan lines). A field named “Headerlength” contains a value of either 8 or 9, where 9 indicates that an optional field appears at the end, called “BitImageFlag”. This field is directly tied with a “NumberOfPlanes” field. If “NumberOfPlanes” is greater than 1, the BitImageFlag indicates whether the image is color or gray-scale. (BitImageFlag = 0 = color, BitImageFlag = 1 = gray scale).

Two fields in the header that are a little unusual among bitmap formats are the width and height of the pixels (in microns), and a field called “PatternLength”. PatternLength is the length of any pattern that will be decoded. It is used by the RLE compression scheme, which assigns one of four different kind of codes to store each of the four different types of repeat data in the image. The “pattern code” indicates that a pattern is available. The pattern and repeat count are decoded while reading the file.

For more on RLE compression, see the ImageGear Supported Compressions Reference section.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

GIF

Full Name	CompuServe Graphics Interchange Format (GIF)
Format ID	IG_FORMAT_GIF = 14
File Extension(s)	*.gif
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

NOTE: To support GIF format ImageGear GIF/TIFF-LZW Component should be attached to Core ImageGear.

ImageGear Supported versions:

- Version 89a, 1989: added ability to store both text and graphics in same file.
- Version 87a, 1987: first version; still read by major applications supporting GIF.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing

ImageGear Read Support:

IG_COMPRESSION_LZW - Indexed RGB: 1, 4, 8 bpp

ImageGear Write Support:

IG_COMPRESSION_LZW - Indexed RGB: 1, 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
VERSION	INT	89	87; 89	Sets version of GIF file to be saved.
INTERLACE	BOOL	FALSE	TRUE, FALSE	Sets GIF interlace mode to be used on saving.

Comments

This is a very popular format for storing graphics images on Web pages. It is also supported by most applications that handle graphical image data.

Unisys, the owner of the LZW compression used on all GIF files, declared at the end of 1995 that it will charge a royalty fee to all developers wishing to use this compression scheme. (This drove many developers to search for a replacement file format. A new file format called PNG (“Ping”) was created. The PNG format is supported by ImageGear; see the PNG section for more information.) However, worldwide GIF-related patents expired in 2004 and the format is once again free to use without the need to pay royalties to Unisys.

While the GIF format is designed to store multiple images, few GIF format viewers support this. For this reason, it is not advisable to store more than one image in a GIF file.

The GIF layout is fairly complex; it can include several categories of “blocks” under which subcategories of blocks may occur.

For both the 89a and 87a versions, the first three blocks are the header, the Logical Source Descriptor, and the Global Color Table. The header simply identifies the file as a GIF and gives the version number. The Logical Source Descriptor is very similar to a header, and is sometimes stored within the header. It contains information about the display screen and color table.

GIFs contain two kinds of color tables: a “Global Color Table” and “Local Color Table.” The Global Color Table is used as a table for the pixel values of all images contained within the GIF file. Optionally, the Local Color Table block that is provided for each image can contain data specific to an individual image.

In addition to a local color table, each image is associated with another block of data, the Local Image Descriptor, provides the size and location of the image, and data about its color table.

GIF89a is equipped with four new types of blocks called “Control Extensions”. The most significant group of these are the “Graphics Control Extension Blocks” that enable the simultaneous storing and displaying of textual and graphical data, including multiple images, resulting in a “multimedia presentation”. The functions provided by these blocks include setting the transparency or opacity of the images, restoring or deleting images, and overlaying captions (that are not part of the actual bitmap) on images.

GIF image data is always stored in LZW-compressed form. The data may also be interlaced. Interlacing helps the appearance of an image as it displays while being decompressed, so that it “fades in”. In a non-interlaced file, the presentation of the image data begins with row 1 and works downward to the last row of data. This method of display does not allow a quick preview of the whole image. When the data is interlaced, the lines are saved and displayed out of sequence. Every fourth row is displayed first and then filled in with every remaining fourth row, until all of the lines are displayed. This allows the eye to perceive the basic subject of the whole image before it is completely displayed.

See also:

GIF Non-image Data Structure

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

Group 3

Full Name	Group 3 (G3)
Format ID	IG_FORMAT_G3 = 11
File Extension(s)	*.raw
Data Type	Raw image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, Unix, Java, Unix64

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
WIDTH	DIMENSION	0	Any positive value	Image width
HEIGHT	DIMENSION	0	Any positive value	Image height
FILL_ORDER	UINT	1		FillOrder = 2, for inverse bit order in byte
PACKED	BOOL	FALSE	TRUE, FALSE	Currently not in use. Reserved for the future.
K_FACTOR	UINT	2		K - factor for 2D coding

Comments

ImageGear does not try to automatically detect raw Group3 data. Use one of the raw file loading functions, or specify format explicitly. Please see the section Loading, Saving and Acquiring Images.

Group 3 2D

Full Name	Group 3 2D
Format ID	IG_FORMAT_G32D = 53
File Extension(s)	*.raw
Data Type	Raw image

Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, Unix, Java, Unix64

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
WIDTH	DIMENSION	0	Any positive value	Image width
HEIGHT	DIMENSION	0	Any positive value	Image height
FILL_ORDER	UINT	1		FillOrder = 2, for inverse bit order in byte
PACKED	BOOL	FALSE	TRUE, FALSE	Currently not in use. Reserved for the future.
K_FACTOR	UINT	2		K - factor for 2D coding

Comments

ImageGear does not try to automatically detect raw Group3 data. Use one of the raw file loading functions, or specify format explicitly. Please see the section Loading, Saving and Acquiring Images.

Group 4

Full Name	Group 4
Format ID	IG_FORMAT_G4 = 12
File Extension(s)	*.raw
Data Type	Raw image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, Unix, Java, Unix64

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
WIDTH	DIMENSION	0	Any positive value	Image width
HEIGHT	DIMENSION	0	Any positive value	Image height
FILL_ORDER	UINT	1		FillOrder = 2, for inverse bit order in byte
PACKED	BOOL	FALSE	TRUE, FALSE	Currently not in use. Reserved for the future.
K_FACTOR	UINT	2		K - factor for 2D coding

Comments

ImageGear does not try to automatically detect raw Group3 data. Use one of the raw file loading functions, or specify format explicitly. Please see the section Loading, Saving and Acquiring Images.

HD Photo

Full Name	HD Photo (formerly Windows Media Photo)
Format ID	IG_FORMAT_HDP = 110
File Extension(s)	*.hdp, *.wdp
Data Type	Raster image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Yes
IG Platforms Support	WIN32 (DLL only), WIN64, .NET, .NET64

NOTE: To support HD Photo format ImageGear HD Photo Component should be attached to Core ImageGear.

ImageGear Supported Versions:

- 1.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_HDP:
  - Grayscale: 1, 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGBA: 32, 64 bpp;
  - RGBPA: 32, 64 bpp;
  - CMYK: 32, 64 bpp;
  - CMYKEx: 40, 80 bpp;
  - Extra: 3…9 channels x 8, 16 bpp;

ImageGear Write Support:

- IG_COMPRESSION_HDP:
  - Grayscale: 1, 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGBA: 32, 64 bpp;
  - RGBPA: 32, 64 bpp;
  - CMYK: 32, 64 bpp;
  - CMYKEx: 40, 80 bpp;
  - Extra: 3…9 channels x 8, 16 bpp;

ImageGear Filter Control Parameters:

Name	Type	Default value	Available values	Description
Lossless	AM_TID_AT_BOOL	FALSE	TRUE, FALSE	Setting this parameter to TRUE enables lossless compression mode.
ImageQuality	AM_TID_DOUBLE	0.9	Any Double value in range 0.0…1.0	Specifies image quality. 0.0 specifies lowest quality, 1.0 specifies highest quality (lossless mode). [1)]

BitmapTransform	AM_TID_AT_MODE	0	One or combination of the following values: 0x00000000 (Rotate 0), 0x00000001 (Rotate 90), 0x00000002 (Rotate 180), 0x00000003 (Rotate 270), 0x00000008 (Flip Horizontal), 0x00000010 (Flip Vertical)	This parameter is stored as the BitMapTransform value in the compressed bit stream header, controlling how the image will be transformed during decode.
UseAdvancedParameters	AM_TID_AT_BOOL	FALSE	TRUE, FALSE	Specifies whether standard or advanced parameters should be used.
Advanced.Quality	AM_TID_BYTE	1	1…255	Controls quantization of the main image. A value of 1 sets lossless mode (no quantization). Increasing values result in greater quantization and thus greater compression ratios and lower image quality. This parameter is tightly related with Advanced.Subsampling and Advanced.Overlap parameters, and its behavior may differ depending on source image content. If the application needs a single parameter to control image quailty, the use of the basic ImageQuality parameter is recommended.
Advanced.Overlap	AM_TID_BYTE	0	0, 1, 2	This parameter selects the optional overlap processing level. [2)]
Advanced.Subsampling	AM_TID_BYTE	3	0, 1, 2, 3	Enables additional compression in the chroma space, preserving luminance detail at the expense of color detail. Only applies to RGB images [3)]
Advanced.HorizontalTileSlices	AM_TID_WORD	0	0…4096	Specifies horizontal tiling of the image. 0 means no horizontal subdividions (1 tile), 1 means 2 horizontal tiles, etc.

Advanced.VerticalTileSlices	AM_TID_WORD	0	0…4096	Specifies vertical tiling of the image. 0 means no vertical subdividions (1 tile), 1 means 2 vertical tiles, etc.
Advanced.FrequencyOrder	AM_TID_AT_BOOL	TRUE	TRUE, FALSE	Specifies that the image must be encoded in frequency order, with the lowest frequency data appearing first in the file, and image content grouped by its frequency rather than its spatial orientation. Organizing a file by frequency order provides the highest performance results for any frequency-based decoding, and is the preferred option.
Advanced.InterleavedAlpha	AM_TID_AT_BOOL	FALSE	TRUE, FALSE	Specifies whether to encode Alpha channel as an additional interleaved channel, or as a planar alpha channel, encoded as a separate image with its own optional Quality value. Interleaved alpha channels are only supported for certain RGB pixel formats. A Planar alpha channel can be associated with any image format that defines an alpha channel.
Advanced.AlphaQuality	AM_TID_BYTE	1	1…255	Specifies quantization for the planar alpha channel image. A value of 1 sets lossless mode (no quantization). Increasing values result in greater quantization and thus greater compression ratios and lower alpha quality.
Advanced.CompressedDomainTranscode	AM_TID_AT_BOOL	TRUE	TRUE, FALSE	Not used
Advanced.ImageDataDiscard	AM_TID_BYTE	0	0…3	Not used
Advanced.AlphaDataDiscard	AM_TID_BYTE	0	0…4	Not used
Advanced.IgnoreOverlap	AM_TID_AT_BOOL	FALSE	TRUE, FALSE	Not used

[1] Quality value maps to specific WMPhoto encoder parameters based on the following table:

ImageQuality	Advanced.Quality (BD<=8)	Advanced.Quality (BD<=16)	Advanced.Quality (BD>16)	Advanced.Subsampling	Advanced.Overlap
>0.0 <0.4	105-IQ*100	185-IQ*180	245-IQ*240	4:4:4	2
>0.4 <1.0	105-IQ*100	185-IQ*180	245-IQ*240	4:4:4	1
1.0	1	1	1	4:4:4	0

[2] Selects the optional overlap processing level

0	No overlap processing is enabled.
1	One level of overlap processing is enabled, modifying 4x4 block encoded values based on values of neighboring blocks.
2	Two levels of overlap processing are enabled; in addition to the first level processing, encoded values of 16x16 macro blocks are modified based on the values of neighboring macro blocks.

[3] Allowed values for Advanced.Subsampling parameter are:

3	4:4:4 encoding preserves full chroma resolution.
2	4:2:2 encoding reduces chroma resolution to ½ of luminance resolution.
1	4:2:0 encoding reduces chroma resolution to ¼ of luminance resolution.
0	4:0:0 encoding discards all chroma content, preserving luminance only. Because the codec uses a slightly modified definition of luminance to improve performance, it is preferred to convert an RGB image to monochrome before encoding rather than use this chroma subsampling mode.

Comments:

HD Photo is a new file format for continuous-tone still images that surpasses the limitations of existing image formats. HD Photo supports a wide range of features including:

- Multiple color formats for display or print
- Fixed or floating point high-dynamic-range image encoding
- Lossless or high-quality lossy compression
- Efficient decoding for multiple resolutions and subregions
- Minimal overhead for format conversion or transformations during decode

HD Photo delivers a lightweight, high performance algorithm with a small memory footprint that enables practical, in-device encoding and decoding. It delivers compression quality comparable with JPEG-2000 and more than twice the quality of JPEG.

HD Photo is the new name for Windows Media Photo. Both names refer to the exact same file format. The Windows Media Photo name is still used to describe the implementation of HD Photo in existing Windows products.

See Also

HD Photo Non-Image Data Structure

References Used

HD Photo v.1.0 Feature Specification.

http://www.microsoft.com/windows/windowsmedia/forpros/wmphoto/default.aspx

HPGL

Full Name	HPGL
Format ID	IG_FORMAT_HPGL = 72
File Extension(s)	*.hpgl, *.plt
Data Type	Vector image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	Win32, Win64, .NET, .NET64

NOTE: To support HPGL format ImageGear CAD/Vector (HPGL, HPGL/2) Component should be attached to Core ImageGear.

ImageGear Supported versions:

HPGL, HPGL/2

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

IG_COMPRESSION_NONE - Vector/CAD data

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
BACKCOLOR	Long	0	Any positive value	This parameter is used to set background color. If nBackGroundColor=0, then a black background color is set, otherwise a white background color is set.
WIDTH	Long	512	Any positive value	Defines the DIB width of the loaded image. The DIB height is calculated using the file's scale factors.
FILENAME	String	""	N/A	Obsolete. Not currently used.

Comments

HPGL ("Hewlett Packard Graphics Language") is a vector graphics language used to output vector/CAD drawings to plotters. Due to widespread support among CAD software, it is sometimes used as a vector/CAD interchange format.

IBM AFP

Full Name	IBM AFP
Format ID	IG_FORMAT_AFP = 106
File Extension(s)	*.afp
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_BW:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_GRAY:
  - Indexed RGB: 4 bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
STITCH_TILES	BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.

Comments:

The AFP Document File is the file, coded in IBM's MO:DCA format (Mixed Object Document Content Architecture). All MO:DCA objects contained in the AFP Document File include the following:

- Document Structure Objects
- Resource Objects
  - Font Objects
  - Overlay Objects
  - Page Segment Objects
- Graphics Objects
  - IM Image Objects
  - Image Objects (IOCA)
  - Graphics Objects (GOCA)
- Text Objects
- Bar Code Objects
- Object Containers

IBM IOCA

Full Name	IBM IOCA (Image Object Content Architecture)
Format ID	IG_FORMAT_ICA =16
File Extension(s)	*.ica, *.mod
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_BW:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_GRAY:
  - Indexed RGB: 4bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
STITCH_TILES	BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.

Comments:

IOCA files are most often used for document storage. They are not so unusual from most file formats, except for their naming conventions, which tend to be IBM-specific.

The general structure of an IOCA image file includes a “beginning segment”, an “end segment”, a header component called “Image Data Parameters”, a palette, and the actual image data. Most IOCA images (less than 24-bit) contain a palette. The elements of the bitmap image are referred to as Image Data Elements (IDEs), that are called pixels by most other formats. The “Object Content” refers to the combination of the header and the image data.

IBM uses fields in the header called “self-defining fields”. They each contain a type code, the length of the parameter, and then the actual parameter data. They include information as resolution, size, encoding scheme, and bit depth.

There are many optional parameters. Subsets of IOCA parameters are referred to as “function sets” and define different flavors of the IOCA, one being the MO:DCA, also supported by ImageGear. Examples of optional parameters include a tiling parameter, if the image is tiled, and a Band Image parameter, which signifies that the image is saved in “bands” (“bit planes” in other formats).

References used

Image Object Content Architecture Reference, 2d ed., copyright International Business Machines Corporation, August 1991.

IBM MO:DCA

Full Name	IBM MO:DCA (Mixed Object Document Content Architecture)
Format ID	IG_FORMAT_MOD = 26
File Extension(s)	*.mod, *.ica
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No

IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_BW:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_GRAY:
  - Indexed RGB: 4bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp

- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
STITCH_TILES	BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.

Comments:

The MO:DCA header allows the storage of multiple IOCA images in one file. The MO:DCA format is an IOCA “wrapper”. It is considered by IBM to be a “data stream controlling environment” for a group of IOCA images.

There are many optional parameters. Subsets of IOCA parameters are referred to as “function sets” and define different flavors of the IOCA, for example, the MO:DCA. The MO:DCA incorporates function sets “10” and “11”.

Please see the description of IBM IOCA for further description.

References used

Image Object Content Architecture Reference, 2d ed., copyright International Business Machines Corporation, August 1991.

ICO

Full Name	ICO (Windows icon)
Format ID	IG_FORMAT_ICO = 17
File Extension(s)	*.ico
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	Single alpha channel for read/write (see Comments).
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Windows 3.1
- Windows NT/95

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Indexed + Extra: 4+1, 8+1 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
READ_AS_RGBA	Boolean	False	True, False	If True, ImageGear reads CUR format as RGBA (RGB with alpha channel).

Comments:

More than one representation of the icon bitmap is stored in order to offer a choice of icons; the version most compatible to the output device is used. Support for read-write transparency masks has been added to the ICO filter. Transparency masks are placed into an alpha channel when the image is created. These masks can subsequently be set and applied to the main image.

The structure of an ICO file consists of four data sections: the header, the Resource Descriptor, and two representations of the image data per each icon (the color bitmap and the 1-bit masking bitmap).

The header identifies the file as an ICO and stores the number of icon images that are stored in the file.

The Resource Descriptor stores the image width and height, the number of colors used, and the offset from the beginning of the file to the image data.

The 1-bit masking bitmap defines the transparent portion of the bitmap.

READ_AS_RGBA control parameter determines how ImageGear reads the 1-bit AND masks. If READ_AS_RGBA is False, ImageGear reads AND mask into “Extra“ channel. This mode preserves unchanged pixel values from the file. However, in this mode ImageGear displays only the XOR mask and ignores AND mask (Extra channel) during display, i.e. display is not transparent. If READ_AS_RGBA is TRUE, ImageGear reads CUR files as 32 bpp RGB + Alpha. This allows transparent display.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

IFF

Full Name	Interchange File Format
Format ID	IG_FORMAT_IFF = 18
File Extension(s)	*.iff
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 1, 1985

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;

- RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

None

Comments

This type of file format is referred to as a “wrapper” because it can include any type of data that is encoded in any manner.

The basic organization of the IFF file format utilizes storage structures known as “chunks”. A chunk is a block of data that contains its own header (that identifies the chunk size and type). This makes it easy for an IFF viewer to identify chunks and to skip over the ones that are not necessary.

The Header Chunk contains 17 fields, including the size of the header chunk, the identification of the chunk as a header, the size and origin of the image, data encoding (yes or no), and aspect ratio.

A “CMG Chunk” may follow, containing data specific to Amiga display hardware.

A “CMAP Chunk” contains the RGB palette for the image.

The “Body Chunk” (also called the ILBM or “interleaved bitmap”), is the image data itself. It is stored in an “interleaved” format, by bit plane. Interleaving allows for data with different resolutions to be neatly stored together. The data may be uncompressed or compressed using an RLE scheme. See RLE section for more information.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

IMG

Full Name	Xerox 9700 graphic image (IMG)
Format ID	IG_FORMAT_XRX = 46
File Extension(s)	*.img
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

Version 1.0

Version 2.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - Indexed RGB: 1, 8 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments

The IMG header contains the width, height and resolution of the image. Following the header is compressed data. The sample values of a binary image are compressed and then encoded into a sequence of bits. Compression is achieved by predicting a pixel value based on pixel values that have already been computed. For example, the predicted value of a pixel may be that of the corresponding pixel on the previous scan line. Up to fifteen different compression techniques are used, each designed to remove redundancy from a certain kind of image - text characters, line art, and halftones of various screen frequencies. The algorithm adapts to the properties of the image by selecting the technique that will perform the best.

References Used

Xerox System Integration Standard. Raster Encoding Standard. Xerox, XNSS 178506, June 1990.

IMR

Full Name	IMRS-Raster image
Format ID	IG_FORMAT_IMR = 59
File Extension(s)	*.ima
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments

The IMG header contains the width, height and resolution of the image. Following the header is compressed data. The sample values of a binary image are compressed and then encoded into a sequence of bits. Compression is achieved by predicting a pixel value based on pixel values that have already been computed. For example, the predicted value of a pixel may be that of the corresponding pixel on the previous scan line. Up to fifteen different compression techniques are used, each designed to remove redundancy from a certain kind of

image - text characters, line art, and halftones of various screen frequencies. The algorithm adapts to the properties of the image by selecting the technique that will perform the best.

References Used

Xerox System Integration Standard. Raster Encoding Standard. Xerox, XNSS 178506, June 1990.

IMT

Full Name	IMNET (Medical Image Format)
Format ID	IG_FORMAT_IMT = 20
File Extension(s)	*.imt
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

NOTE: The width of an image being saved must be a multiple of 8.

ImageGear Filter Control Parameters:

None

JBIG2

Full Name	JBIG2 (bilevel image compression)
Format ID	IG_FORMAT_JB2 = 62
File Extension(s)	*.jb2
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32

NOTE: To support JBIG 2 format ImageGear JBIG2 Component should be attached to Core ImageGear.

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing

ImageGear Read Support:

IG_COMPRESSION_JBIG2 - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_JBIG2 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Please see the section JBIG2 Filter Control Parameters.

Comments

JBIG2 is a highly-compressed black and white image format that uses symbol recognition and substitution for very good compression results. Any black and white image can be compressed using JBIG2 including images stored as Group 4, Group 3, MO:DCA, TIFF, PDF and others.

The JBIG2 compression algorithm searches for groups of pixels arranged in similar shapes and use them to define a symbol. Rather than retaining information on the placement of each pixel, a table is created for common symbols. This allows users to see incredible reductions in file sizes. Compression rate matches 100:1 compared to uncompressed black and white TIFFs.

JPEG

Full Name	JPEG File Interchange Format
Format ID	IG_FORMAT_JPG = 21
File Extension(s)	*.ipg, *jpeg
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Yes
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC, Java

ImageGear Supported versions:

- Version 1.01 – 1991
- Version 1.02 - Added ability for thumbnails to be color-mapped and JPEG compressed.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_JPEG (Lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - CMYK: 32 bpp;

        - RGB + Alpha: 32 bpp
    - IG_COMPRESSION_JPEG (Lossless):
        - Grayscale: 8, 16 bpp;
        - RGB: 24, 48 bpp;
        - CMYK: 32 bpp;
        - RGB + Alpha: 32 bpp
    - IG_COMPRESSION_JPEG (Progressive JPEG):
        - Grayscale: 8, 12 bpp;
        - RGB: 24 bpp;
        - CMYK: 32 bpp;
        - RGB + Alpha: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_JPEG (Lossy):
    - Grayscale: 8, 12 bpp;
    - RGB: 24, 36 bpp;
    - CMYK: 32 bpp;
- IG_COMPRESSION_JPEG (Lossless):
    - Grayscale: 8, 16 bpp;
    - RGB: 24 bpp;
- IG_COMPRESSION_JPEG (Progressive JPEG):
    - Grayscale: 8, 12 bpp;
    - RGB: 24, 36 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
CALLER_ID	Long	21		For internal use.

Filter Control Parameter	Type	Default Value	Available Values	Description
LOAD_SAVE_DCT	BOOL	FALSE	FALSE TRUE	For internal use. Load/save DCT matrix instead of decompressing/ compressing the image
QUALITY	INT	70	Possible value in range [1,100]	Gets/Sets quality value for Lossy and Progressive JPEG compression. See JPEG Quality Section for details.
DECIMATION_TYPE	WORD	IG_JPG_DCM_2x2_1x1_1x1	See JPEG Decimation Section.	Get/Set decimation value. For all available values see JPEG Decimation Section.
SAVE_THUMBNAIL	BOOL	FALSE	FALSE TRUE	Gets/Sets thumbnail flag. If TRUE then thumbnail will be added to image. See JPEG Decimation Section for details.
ENTROPY_OPTIMIZE	BOOL	FALSE	FALSE TRUE	Get/Set entropy optimization flag. See JPEG Entropy Optimization Section.
OLD_LOSSLESS_READ	BOOL	FALSE	FALSE TRUE	
THUMBNAIL_WIDTH	UINT	16		Gets/Sets thumbnail width.
THUMBNAIL_HEIGHT	UINT	16		Gets/Sets thumbnail height.
PREDICTOR	AT_MODE	1		Gets/Sets predictor for lossless JPEG. See Lossless JPEG Predictor Settings Section for details.

Filter Control Parameter	Type	Default Value	Available Values	Description
SCAN_INFO_COUNT	UINT	sizeof(scan_info)/ sizeof(scan_info[0]) ;		Gets/Sets length of array for previous parameter. See Progressive JPEG Scans Section for details.
SCAN_INFO	VOID|LP	{ { 0, 0, 7, 1, 0 }, { 1, 5, 7, 1, 1 }, { 1, 5, 7, 1, 2 }, { 1, 5, 7, 1, 3 }, { 0, 0, 0, 0, 0 }, { 6, 63,7, 1, 1 }, { 6, 63,7, 1, 2 }, { 6, 63,7, 1, 3 }, { 1, 63,0, 0, 1 }, { 1, 63,0, 0, 2 }, { 1, 63,0, 0, 3 }, };		Gets/Sets scan configuration for Progressive JPEG format. See Progressive JPEG Scans Section for details.
LOAD_SCANS	UINT	0		Gets/Sets number of scans to load from Progressive JPEG image. See Progressive JPEG Scans Section for details.
KEEP_ALPHA	BOOL	FALSE	FALSE TRUE	This parameter gets/sets alpha channel flag. If TRUE then alpha channel will be saved with original image (if it is present).

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_TYPE	AT_MODE	IG_JPG_LOSSY	IG_JPG_LOSSY, IG_JPG_LOSSLESS, IG_JPG_PROGRESSIVE	Get/Set type for output JPEG format. See Loading and Saving JPEG-Compressed Images Section for details.
LOAD_SCALE_DENOM	UINT	1	1, 2, 4, 8	If this parameter is set to any other than default value, ImageGear loads reduced version of the image, width and height of which are scaled by 1/ load_scale_denom. This mode can be used for image preview, especially for those images that do not have embedded thumbnails, or where embedded thumbnails are smaller than desired. This mode allows you to make loading process 2-4 times faster.*
SAVE_ALLOW_CMYK	BOOL	TRUE	TRUE, FALSE	Set to TRUE (default) to allow saving JPEG CMYK images. Set to FALSE to save CMYK images to JPEG as RGB.

Comments:

JPEG is normally associated with the JPEG compression scheme, but it is also implemented into the JFIF file format. This format was developed to store JPEG-encoded data, and to exchange it between applications or operating systems that are normally incompatible.

NOTE: The JPEG compression scheme was developed by the Joint Photographic Experts Group (created by the joining of a subgroup of the International Standards Organizations, called PEG (Photographic Experts Group) and a subgroup of the CCITT). Their common goal was to produce a standard for the transmission of graphics image data over networks and through color facsimile systems.

The header of the JFIF contains the version number, the image dots per inch (DPI), or dots per centimeter, and an optional thumbnail (miniature) RGB representation of the main image. Version 1.02 handles thumbnails differently by storing them separately, rather than in the identification marker of the header.

The raw JPEG data is surrounded by two markers, an “SOI” (start of image) marker, and an “EOI” (end of image) marker. See the section entitled Compression Schemes for more about JPEG compression.

JFIF is considered a non-proprietary file format. Many proprietary file formats contain JPEG data, incorporating their own application-specific data structures. Other non-proprietary formats that use JPEG-encoded data are Macintosh MAC PICT and the TIFF file format, version 6.0.

Loading and Saving JPEG-Compressed Images

ImageGear supports the reading and writing of three types of JPEG compression: baseline JPEG (**Lossy**), **Progressive** JPEG, and **Lossless** JPEG. When you load a JPEG-compressed file, ImageGear detects the type of JPEG comp

ression and decompress the image automatically. But if you want to save an image with a JPEG compression scheme other than baseline JPEG, you must use the “SAVE_TYPE” parameter to specify the type of JPEG compression.

Use these constants for “SAVE_TYPE” control parameter:

- IG_JPG_LOSSY - Lossy JPEG compression.
- IG_JPG_LOSSLESS - Lossless JPEG compression
- IG_JPG_PROGRESSIVE - Progressive JPEG compression.

JPEG Decimation

This table lists all possible decimation values:

IG_JPG_DCM_1x1_1x1_1x1	IG_JPG_DCM_2x1_1x1_1x1
IG_JPG_DCM_1x2_1x1_1x1	IG_JPG_DCM_2x2_1x1_1x1
IG_JPG_DCM_2x2_2x1_2x1	IG_JPG_DCM_4x2_1x1_1x1
IG_JPG_DCM_2x4_1x1_1x1	IG_JPG_DCM_4x1_1x1_1x1
IG_JPG_DCM_1x4_1x1_1x1	IG_JPG_DCM_4x1_2x1_2x1
IG_JPG_DCM_1x4_1x2_1x2	IG_JPG_DCM_4x4_2x2_2x2

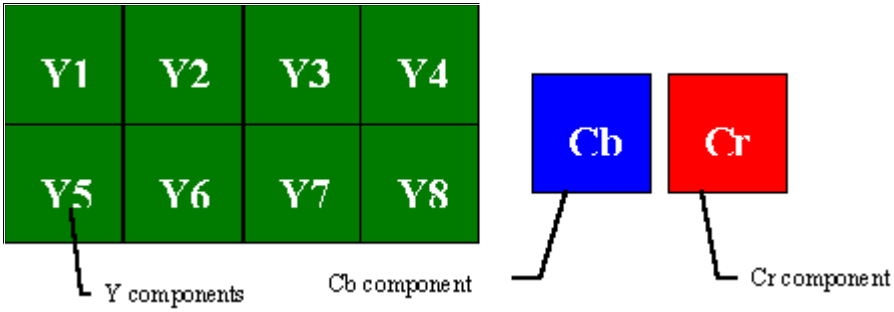
The format of these ImageGear decimation constants is:

IG_JPG_DCM_<H1>x<V1>_<H2>x<V2>_<H3>x<V3>,

where Hi, Vi = horizontal and vertical decimation values for the i-channel.

The following is a simple example of decimation. For a more detailed definition, please see the JPEG Specification.

A decimation setting of IG_JPG_DCM_4x2_1x1_1x1 would yield the following results:



As shown, 8 Y components in the source image have yielded one Cb and one Cr component. In general, this setting will reduce the quality of the compression, unless the image has many continuous tone areas.

- Maximum quality can be reached using a value of IG_JPG_DCM_1x1_1x1_1x1.
- Maximum compression ratio can be reached with a value of IG_JPG_DCM_4x4_2x2_2x2.
- The ImageGear default decimation value is: IG_JPG_DCM_2x2_1x1_1x1.

JPEG Thumbnails

The JPEG format can store thumbnails, which are small representations of the original image. These images are stored in uncompressed form and can significantly decrease your overall compression ratio. (Uncompressed thumbnails sometimes occupy more space than the original JPEG image when compressed). Use this option carefully.

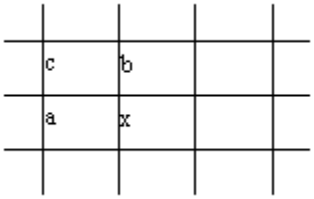
NOTE: The JPEG format does not allow the storage of “large” thumbnails. This is due to the marker segment length, which cannot be greater than 65,536 bytes. The maximum size of a color thumbnail is about 100x200 or 200x100 pixels, and the maximum size of a gray-scale thumbnail is about 300x200 or 200x300 pixels.

Lossless JPEG Predictor Settings

The Lossless JPEG scheme is “predictive” in nature—it uses the values of surrounding pixels in addition to the value of the original pixel to calculate a predictor value, which it then subtracts from the value of the original pixel. The resulting pixel value will be reduced such that it can be compressed more than the original value. The higher the number of neighboring pixels used, the higher the compression will be.

Regardless of the predictor value setting, the quality of the image will remain the same. The difference is that if you choose to optimize for space by setting a high predictor value, you will have to give up some speed, as the decompression will take longer to perform.

ImageGear lets you set the predictor value using `"PREDICTOR"` control parameter. The ImageGear default for this setting is 1. The allowed range is 1—7. The graphic below shows a predictor (x) and three reconstructed samples (a,b,c) immediately to the left, immediately above, and diagonally to the left of the current sample



Lossless JPEG does not apply DCT for an image as per the Lossy JPEG compression. Instead, it uses a DPCM difference coding, which can be carried out with any one of seven different prediction modes. Correspondingly, the `IG_CONTROL_JPG_PREDICTOR` control parameter can be set to a value between 0 and 7. In Table 4, you will see what algorithm your setting of 1—7 will use, where Pr Px is the predictor, and Ra, Rb, and Rc are the reconstructed samples:

Value	JPEG DIS Prediction
0	no prediction
1	Px = Ra
2	Px = Rb
3	Px = Rc
4	Px = Ra + Rb – Rc
5	Px = Ra + ((Rb-Rc)/2)
6	Px = Rb + ((Ra-Rc)/2)
7	Px = (Ra + Rb)/2

JPEG Quality

The baseline JPEG specification calls for a quality setting. The lower the setting, the greater the number of original pixels lost, and therefore the smaller the resulting compressed file will be. ImageGear lets you set the quality of compression with values of type INT between 1-100, where 100 provides the highest retention of original pixel values. A setting of 100 does not mean that the image includes 100% of all original pixel values. With Lossy JPEG, there is no such thing as “no loss”. Control parameter for setting Lossy JPEG quality is `"QUALITY"`.

JPEG Entropy Optimization

Entropy optimization is relevant only to standard Lossy JPEG compression. If you set the parameter to **TRUE**, the default Huffman tables are not used. Instead, optimal Huffman tables are created for each component. This can bring a higher compression ratio but it takes more time for the compression.

Progressive JPEG Scans

A Progressive JPEG file stores more than just a copy of an image but rather several scans, each of which progressively adds a higher level of quality. Each scan contains a portion of the original image data. The purpose of this is to allow a very quick display of an image, beginning with a low-quality rendering and then increasing in quality as the remaining scans are added to it.

A Progressive JPEG image is stored as sequence of Huffman compressed blocks or “scans”. Each scan contains the sequence of DCT coefficients in the given range. However, the coefficients are not complete. Only some of their bits will be stored in each scan.

ImageGear defines the following structure for holding the necessary configuration to write a JPEG image:

```
typedef struct tag AT_PJPEG_SCANINFO{
       LONG   Ss;
       LONG   Se;
       LONG   HBit;
       LONG   LBit;
       LONG   ChannelID;
}      AT_PJPEG_SCANINFO;
typedef AT_PJPEG_SCANINFO FAR* LPAT_PJPEG_SCANINFO;
```

The **Ss** and **Se** members of the `AT_PJPEG_SCANINFO` structure are used for spectral selection control coefficients:

```
LONG      Ss;
LONG      Se; - after applying DCT of 8x8 pixels we get 64
```

In Progressive coding, these coefficients are separated into different scans. Values **Ss** and **Se** specify the first and last number of the DCT coefficients that must be included in a given scan. The possible values are Ss = Se = 0 or 1<= Ss<=Se<=63. Please note following restrictions:

- The first coefficient (DC) cannot be encoded with the other coefficient (AC) in the single scan. In other words, the DC and AC coefficients cannot be in the same scan.
- Only scans that code DC coefficients may include interleaved blocks from more than one component. All other scans shall have only one component. For each component, a first DC scan shall precede any AC scans.

The `HBit` and `LBit` members of the `AT_PJPEG_SCANINFO` structure are used for successive approximation control:

```
LONG HBit;
LONG LBit;
```

If successive approximation is used, the DCT coefficients are reduced in precision by the point transform defined in the scan header. This is equivalent to taking some binary digits from each coefficient. `HBit` and `LBit` specify the high and low range of bits to take. For example, if `HBit` = 7 and `LBit` = 2, the scan will have the following original bits of the original DCT coefficient: 7,6,5,4,3,2.

The `ChannelID` member of the `AT_PJPEG_SCANINFO` structure is used to specify the number of components that will be encoded:

```
LONG ChannelID;
```

It can be set to one of the following values:

- interleaved scan which will only have DC coefficients of all components. This setting can only be used if Ss=Se=0.
- takes coefficients of first component.
- takes coefficients of second component.
- takes coefficients of third component.

There are two control parameters that operate with the `AT_PJPEG_SCANINFO` structure:

`SCAN_INFO`	Points to array of `AT_PJPEG_SCANINFO` elements.
`SCAN_INFO_COUNT`	Specifies the number of elements in this array. The nth entry of the `SCAN_INFO` array defines the configuration for the nth scan of the Progressive image.

The `LOAD_SCANS` control parameter specifies how many scans should be loaded. For example, if it is set to 1, the JPEG filter will load only the first scan of the image.

The following text blocks represent the different `AT_PJPEG_SCANINFO` structures that would be generated when loading a JPEG file using the ImageGear default settings for Progressive scans:

```
scan_config[0].Ss = 0;
scan_config[0].Se = 0;
scan_config[0].HBit = 7;
scan_config[0].LBit = 2;
scan_config[0].ChannelID = 0;
```

```
scan_config[1].Ss = 1;
scan_config[1].Se = 5;
scan_config[1].HBit = 7;
scan_config[1].LBit = 2;
scan_config[1].ChannelID = 1;
```

```
scan_config[2].Ss = 1;
scan_config[2].Se = 5;
scan_config[2].HBit = 7;
scan_config[2].LBit = 2;
scan_config[2].ChannelID = 2;
```

```
scan_config[3].Ss = 1;
scan_config[3].Se = 5;
scan_config[3].HBit = 7;
scan_config[3].LBit = 2;
scan_config[3].ChannelID = 3;
```

```
scan_config[4].Ss = 0;
```

```
scan_config[5].Ss = 1;
```

```
scan_config[4].Se = 0;
scan_config[4].HBit = 1;
scan_config[4].LBit = 1;
scan_config[4].ChannelID = 0;

scan_config[5].Se = 5;
scan_config[5].HBit = 1;
scan_config[5].LBit = 1;
scan_config[5].ChannelID = 1;

scan_config[6].Ss = 1;
scan_config[6].Se = 5;
scan_config[6].HBit = 1;
scan_config[6].LBit = 1;
scan_config[6].ChannelID = 2;

scan_config[7].Ss = 1;
scan_config[7].Se = 5;
scan_config[7].HBit = 1;
scan_config[7].LBit = 1;
scan_config[7].ChannelID = 3;

scan_config[8].Ss = 6;
scan_config[8].Se = 63;
scan_config[8].HBit = 7;
scan_config[8].LBit = 1;
scan_config[8].ChannelID = 1;

scan_config[9].Ss = 6;
scan_config[9].Se = 63;
scan_config[9].HBit = 7;
scan_config[9].LBit = 1;
scan_config[9].ChannelID = 2;

scan_config[10].Ss = 6;
scan_config[10].Se = 63;
scan_config[10].HBit = 7;
scan_config[10].LBit = 1;
scan_config[10].ChannelID = 3;

scan_config[11].Ss = 0;
scan_config[11].Se = 0;
scan_config[11].HBit = 0;
scan_config[11].LBit = 0;
scan_config[11].ChannelID = 0;

scan_config[12].Ss = 1;
scan_config[12].Se = 63;
scan_config[12].HBit = 0;
scan_config[12].LBit = 0;
scan_config[12].ChannelID = 1;

scan_config[13].Ss = 1;
scan_config[13].Se = 63;
scan_config[13].HBit = 0;
scan_config[13].LBit = 0;
scan_config[13].ChannelID = 2;

scan_config[14].Ss = 1;
scan_config[14].Se = 63;
scan_config[14].HBit = 0;
scan_config[14].LBit = 0;
scan_config[14].ChannelID = 3;
```

References used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

JPEG2000

Full Name	JPEG2000
Format ID	IG_FORMAT_JPEG2K = 100
File Extension(s)	*.jp2, *.jpc, *.j2k, *.j2c
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Read/write
IG Platforms Support	WIN32, WIN64, .NET, .NET64

NOTE: To support JPEG2000 format ImageGear JPEG2000 Component should be attached to Core ImageGear.

ImageGear Supported versions:

ISO/IEC 15444-1:2000

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1..8 bpp;
  - Grayscale: 1..16 bpc
  - RGB: 1..16 bpc
  - Grayscale + Alpha: 1..16 bpc

- RGB + Alpha: 1..16 bpc

ImageGear Write Support:

- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1..8 bpp;
  - Grayscale: 1..16 bpc
  - RGB: 1..16 bpc
  - Grayscale + Alpha: 1..16 bpc
  - RGB + Alpha: 1..16 bpc

ImageGear Filter Control Parameters:

Please see the section JPEG2000 Format Support and JP2, JPX and MJ2 Format Filters Control Parameters Reference.

Comments:

JPEG 2000 standard defines a compression scheme and a file format for interchange of JPEG2000 compressed data.

The compression scheme is based on a discrete wavelet transform, and supports both lossy and lossless modes. JPEG 2000 compression is similar to JPEG in that it supports lossy compression. However, JPEG2000 allows achieving much higher compression ratios while preserving good image quality.

JPEG2000 file format defines a basic set of features that allow interchange of JPEG2000 compressed images.

JPX

Full Name	JPEG2000 Extensions
Format ID	IG_FORMAT_JPX = 101
File Extension(s)	*.jpx
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Yes
IG Platforms Support	WIN32, WIN64, .NET, .NET64

NOTE: To support JPX format ImageGear JPEG2000 Component should be attached to Core ImageGear.

ImageGear Supported versions:

ISO/IEC 15444-2:2000

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1..8 bpp;
  - Grayscale: 1..16 bpc
  - RGB: 1..16 bpc
  - YCbCr: 1..16 bpc
  - Lab: 1..16 bpc
  - CMY: 1..16 bpc
  - CMYK: 1..16 bpc
  - Grayscale + Alpha: 1..16 bpc
  - RGB + Alpha: 1..16 bpc

ImageGear Write Support:

- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1..8 bpp;
  - Grayscale: 1..16 bpc
  - RGB: 1..16 bpc
  - YCbCr: 1..16 bpc
  - Lab: 1..16 bpc
  - CMY: 1..16 bpc
  - CMYK: 1..16 bpc
  - Grayscale + Alpha: 1..16 bpc
  - RGB + Alpha: 1..16 bpc

ImageGear Filter Control Parameters:

Please see the section JPX Format Support and JP2, JPX and MJ2 Format Filters Control Parameters Reference.

Comments:

JPX file format is an extension to JPEG 2000 file format. It normally uses JPEG 2000 compression, but allows other compressions as well, such as JBIG, CCITT Group3, CCITT Group4, etc. The format provides advanced features such as multiple layers for an image, various colorspaces, opacity (alpha channel) and fragmented codestreams.

KFX

Full Name	KFX (Kofax Group4 image)
Format ID	IG_FORMAT_KFX = 22
File Extension(s)	*.kfx
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 3

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G4 - 1bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

LV

Full Name	LV (Lazer View format)
Format ID	IG_FORMAT_LV = 23
File Extension(s)	*.lv
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G3 - 1bpp
- IG_COMPRESSION_CCITT_G4 - 1bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments:

N/A

MAC

Full Name	MAC (Macintosh Paint)
Format ID	IG_FORMAT_MAC = 24

File Extension(s)	*.mac
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, MAC

ImageGear Supported versions:

Version 2.0 1989

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_RLE - 1 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments:

Originally developed to store MacPaint graphics files, this format is now supported by many Macintosh applications. It is also exportable to the PC platform. It is always monochrome, and always has a fixed size of 576 pixels by 720 lines. The data, when uncompressed, is always 51,840 bytes in size.

Because this is a Macintosh format, it is organized as “forked” data. Each file consists of two forks, a “resource fork” and a “data fork”. There is no code associated with this graphics format. The resource fork is always empty, and is easily merged together with the data fork when the file is exported to a PC platform.

The MacPaint data begins with a version number. If set to a value of 2, it indicates that paint patterns appear as the next structure. There are 38 possible patterns. These are generally not used, unless the file is being exported from one paint program to another.

The bitmap data begins at an offset of 512 bytes from the beginning of the file. The data is always compressed using the “PackBits” RLE compression scheme. The compressed data is stored in variable-length strips. See the description of RLE compression in the ImageGear Supported Compressions Reference section.

It the MacPaint file has been exported to a PC platform it contains a structure called the MacBinary header. This helps in reconstructing the resource fork if the file is returned to the Macintosh environment. A field in the MacBinary header

holds the size of the fork. Other information includes the position of the file in the window, the version of the MacBinary header (I or II), the time and data of creation, and a SecondHeadLength field intended for future expansion of the MacPaint format should it require a secondary header.

References used

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

MAC PICT

Full Name	Mac PICT
Format ID	IG_FORMAT_PCT = 30
File Extension(s)	*.pct, *.pict
Data Type	Metafile (2D raster, 2D geometry)
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Supports single alpha channel for read only. Alpha channel have to be 8-bit image.
IG Platforms Support	WIN32, WIN64, MAC

ImageGear Supported versions:

- PICT 2 (for Color QuickDraw version 2). Added color and support for additional QuickDraw functions.
- PICT 1 (for Color QuickDraw version 1). Monochrome only.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp and 24+8 bit alpha

- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp and 24+8 bit alpha
- IG_COMPRESSION_JPEG:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp and 24+8 bit alpha

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp (for 1, 4, 8 bpp, the files are ONLY saved as uncompressed if the image is 64x64 pixels or smaller. Otherwise, the image is saved as RLE compressed).
  - RGB: 24 bpp (Always saved as uncompressed)

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
FORCE_VERSION	BOOL	FALSE	TRUE, FALSE	TRUE means saving of PCT in version 1.

Comments:

The PICT is one of the most widely-supported graphics file formats for the Macintosh.

PICT files begin with a fixed length header containing application-specific data, followed by fields that store the image size and location. If it is a PICT2 file, an additional header follows that contains the original resolution data of the image.

The bitmap data in the PICT2 format is referred as a Pixmap, from older terminology where Pixmap meant a bitmap with color.

The PICT file uses “opcodes”. These are similar to the fields found in most file formats, and are associated with data that describes different shapes, lines, fill patterns, etc.

The lengthy list of opcodes is followed by the bitmap or “pixmap” data that describes the image data’s address and resolution. The color table follows the opcodes. Next, source and destination rectangles are defined by their top left and lower right coordinates.

The pixel data, stored in the “PixData” field, is the last data to appear in the file. Each value is an index to the color table. This data is represented by 1, 2, or 4 bits.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

Motion JPEG2000

Full Name	Motion JPEG2000
Format ID	IG_FORMAT_MJ2 = 103
File Extension(s)	*.mj2
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Read only
IG Platforms Support	WIN32, WIN64, .NET, .NET64

NOTE: To support Motion JPEG2000 format ImageGear JPEG2000 Component should be attached to Core ImageGear.

ImageGear Supported versions:

ISO/IEC 15444-3:2002

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_JPEG2K:
  - Indexed RGB: 1..8 bpp;
  - Grayscale: 1..16 bpc
  - RGB: 1..16 bpc
  - Grayscale + Alpha: 1..16 bpc

- RGB + Alpha: 1..16 bpc

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Please see the section MJ2 Format Support and JP2, JPX and MJ2 Format Filters Control Parameters Reference.

Comments:

Motion JPEG2000 file format is designed to contain one or more motion sequences of JPEG2000 images, with their timing, and also optional audio annotations, all composed into an overall presentation.

Please see the section MJ2 Format Support for more details.

MSP

Full Name	MSP (MS Paint)
Format ID	IG_FORMAT_MSP = 25
File Extension(s)	*.msp
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 2.0
- Version 1.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE: Indexed RGB: 1 bpp
- IG_COMPRESSION_RLE: Indexed RGB: 1 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments:

This was a popular format for storing line drawing and clip art images created with Windows applications, but has recently begun to be replaced by the Windows bitmap format (also supported by ImageGear see BMP).

For versions earlier than 2.0, the format begins with a 32-byte header and immediately proceeds with the bitmap data. The header information includes: the version of the file, the size and aspect ratio of the bitmap, and the width and height and aspect ratio (in pixels) of the output device used to render the bitmapped image. The bitmap data is uncompressed.

Files made with Version 2.0 and later always use RLE compression. In these files a “scan-line map” follows the header and precedes the data. It gives offsets to each scan line for instances when a particular scan line needs to be examined. All previous lines can remain compressed, while the needed line is located.

See RLE Compression under the ImageGear Supported Compressions Reference section for more information.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

MULTIMEDIA

Full Name	Multimedia, Motion Picture Expert Group (MPEG)
Format ID	IG_FORMAT_MUL = 67
File Extension(s)	*.avi, *.mpeg, *.mpg
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64

NOTE: To support MULTIMEDIA format ImageGear Multimedia Component should be attached to Core ImageGear.

ImageGear Supported versions:

- AVI
- MPEG-1 (Standard)
- MPEG-2
- MPEG-3
- MPEG-4

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - RGB: 24 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
FILENAME	LPSTR	“”	Any string	Name of the input MPEG or AVI file.

Comments:

The first version of the MPEG format is optimized for CD-ROM. It uses discrete cosine transform (DCT) and Huffman compression to remove spatially redundant data within a frame and block-based motion compensated prediction (MCP) to remove data which is temporally redundant between frames. Audio is compressed using subband compression.

MPEG-2 is a variant of the MPEG video and audio compression algorithm and file format, optimized for broadcast quality video for digital storage media up to 4.0Mbits/second. The file extension is .MP2. MPEG-2 has been approved as International Standard IS-13818.

MPEG-3 is a variant of the MPEG video and audio compression algorithm and file format. The file extension is .MP3. This variant no longer exists and has been merged into MPEG-2.

MPEG-4 is a variant of the MPEG video and audio compression algorithm and file format used for low bandwidth video telephony. The file extension is .MP4. The International Standards Organization (ISO) has adopted the QuickTime 3 file format to form the starting point for a unified digital media storage format for the MPEG-4 specification

No actual structured MPEG file format has been defined. Everything required to play back MPEG data is compressed directly in the data stream. Therefore, no header or other type of wrapper is necessary. It is likely that when needed, a multimedia standards committee - perhaps MHEG or the DSM (Digital Storage Medium) MPEG subgroup - will one day define an MPEG file format.

MPEG starts with a relatively low resolution video sequence (possibly decimated from the original) of about 352 by 240 frames by 30 frames in the US (the numbers are different in Europe), but original high (CD) quality audio. The images are in color, but converted to YUV space, and the two chrominance channels (U and V) are decimated further to 176 by 120 pixels. You can get us much less resolution in those channels and not notice it, at least in natural (not computer generated) images.

The basic scheme is to predict motion from frame to frame in the temporal direction, and then to use DCTs (Discrete Cosine Transforms) to organize the redundancy in the spatial directions. The DCTs are performed on 8x8 blocks. The motion prediction is performed in the luminance (Y) channel on 16x16 blocks. In other words, given the 16x16 block in the current frame that you are trying to compress, you look for a close match to that block in a previous or future frame (there are backward prediction modes where later frames are sent first to allow interpolating between frames). The DCT coefficient (of either the actual data, or the difference between this block and the close match) are quantized, which means that you divide them by some value to drop bits off the bottom end. Hopefully, many of the coefficients will then end up being zero. The quantization can change for every macroblock (a macroblock is 16x16 of Y and the corresponding 8x8's in both U and V). The results of all of this, which include the DCT coefficients, the motion vectors, and the quantization parameters (and other stuff) is Huffman compressed using fixed tables. The DCT coefficients have a special Huffman table that is two-dimensional in that one code specifies a run-length of zeros and the non-zero value that ends the run. Also, the motion vectors and the DCT components are DPCM (subtracted from the last one) compressed.

There are three types of compressed frames.

- **I** or **intra** frames are simply a frame compressed as a still image, not using any past history.
- **P** or **predicted** frames are predicted from the most recently reconstructed I or P frame. (Described from the decompressor view point) Each macroblock in a P frame can either come with a vector and difference DCT coefficients for a close match in the last I or P, or it can just be intra compressed (like in the I frames) if there is no good match.
- **B** or **bidirectional** frames are predicted from the closest two I or P frames, one in the past and one in the future. You search for matching blocks in those frames, and try three different things to see which works best. (Described from the compressor view point.) You first use the forward vector and the backward vector, then average the two blocks from future and past frames,

and then subtract that from the block being compressed. If none of these methods work well, you can intra-compress the block.

The sequence of decompressed frames is as follows:

```
IBBPBBPBBPBBIBBPBBPB...
```

Where there are 12 frames from I to I (for US and Japan). This is based on a random access requirement that you need a starting point; at least once every 0.4 seconds or so. The ratio of P's to B's is based on experience. Of course, for the decompressor to work, you have to send that first P before the first two B's, so that the compressed data stream appears as follows:

```
0xx312645...
```

where those are frame numbers. “xx” may be nothing (if this is the true starting point), or it may be the B's of frames -2 and -1 if it is the middle of the stream.

Decompress the I, then the P. Keep both in memory, and then decompress the two B's. You can display the I while you're decompressing the P, and display the B's as you're decompressing them. Display the P as you are decompressing the next P, and so on.

See also AVI file format reference.

NCR

Full Name	NCR
Format ID	IG_FORMAT_NCR = 27
File Extension(s)	*.ncr
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

- IG_COMPRESSION_NONE - Indexed RGB: 1, 4 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
COMRESSION	WORD	0	0, NCR_CCITT_G4	This parameter has been retired.

Comments:

NCR is the black and white image compression format.

PBM

Full Name	PBM (Portable Bitmap File Format)
Format ID	IG_FORMAT_PBM = 28
File Extension(s)	*.pbm
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW:

- Indexed RGB: 1 bpp;
- Grayscale: 8, 16 bpp;
- RGB: 24, 48 bpp

- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Filter Control Parameters:

None

Comments:

This is useful for quick and easy transfer of monochrome bitmap images, although the encoding scheme is not efficient in terms of storage space. This format, as well as the PGM, PNM, and PPM formats, are at the core of a set of utility programs also written by Jef Poskanzer. Among other things, these formats serve as intermediary storage methods for the conversion of other file formats.

The Portable Bitmap File Format structure is very simple. It begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, and perhaps a comment line identifying the filename. Following white space (usually a carriage return) is the bitmap data. The number of bits is equal to the width * height. A pixel value of 0 indicates white, and a value of 1 indicates black.

The magic number of the header can have one of two values: either P1 or P4. P1 indicates that the bitmap data are to be read as ASCII decimal values. P4 indicates that the bitmap data are stored as plain bytes. Because 8 pixel values (1 bit each) are stored in one byte, the file is 8 times smaller than in the ASCII decimal format. White spaces are permitted in the P1 format but not permitted in the P4 format.

NOTE: Note: ImageGear uses `IG_FORMAT_PBM` filter to handle the whole family of formats: PBM (1-bit), PGM (grayscale), PPM (truecolor) and PNM (collective name for all of above). When saving image as `IG_FORMAT_PBM` format, ImageGear chooses particular format (PBM, PGM, PNM or PPM) depending on image bit depth.

See Also:

PGM, PPM, PNM

References used

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest / McGraw-Hill, 1995.

PBM Specification by Jef Poskanzer, copyright © 1989, 1991.

PCD

Full Name	PCD (Kodak Photo CD)
Format ID	IG_FORMAT_PCD = 29
File Extension(s)	*.pcd
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

All valid PCD files in 5 different resolutions:

- Page 1: 768x512
- Page 2: 384x256
- Page 3: 192x128
- Page 4: 1536x1024
- Page 5: 3072x2048

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - RGB: 24 bpp
- IG_COMPRESSION_HUFFMAN - RGB: 24 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments:

"Photo CD" is the informally adopted name for files created using the Photo CD-ROM-based storage and retrieval system created by Kodak. The images are digitized versions of photographic images. Using a Photo CD player, images can be viewed on television. Although intended for photographic images, the data source does not necessarily have to be film. Due to the large storage capacity of the CD medium, this format supports very large and/or intricate images.

Images and their associated information are stored in groups called "sessions". Originally, they were stored at the rate of one session per CD, but later versions allowed multiple sessions per disc.

For each stored image, there are up to 5 bitmaps, each representing the image at a different resolution. The bitmaps at the lowest resolution are intended for such purposes as displaying thumbnails and previewing an image.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

PCX

Full Name	PCX (PC Paintbrush File Format)
Format ID	IG_FORMAT_PCX = 31
File Extension(s)	*.pcx
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No

IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 5.0 1991, Supports 24-bit RGB
- Version 3.0
- Version 2.8 Included color palette
- Version 2.5

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_COMPRESSED	BOOL	TRUE	FALSE, TRUE	If TRUE - image is saved with RLE Compression

Comments:

PCX is one of the oldest PC-based bitmap formats. It became well-known when Microsoft used it for their Paintbrush for Windows application and distributed it with every copy of Windows. This format is popular for fax documents because it

allows them to be viewed within many popular paint and image display programs. If multiple images are desired in a PCX format, the format known as DCX, designed for this purpose, (and supported by ImageGear), may be used. Please see DCX file format for more information.

The main components of the PCX file format are the fixed length header, the image data, and if it is written for VGA display technology, the palette for the image (this appears as the last structure in the file). The header includes fields including the PCX version, the image size, resolution and position, and an encoding field that always has a value of 1; PCX data is always RLE. (For complex images, this may actually cause the bitmap data to increase in size).

The size and location of the palette associated with the image depends on the version of the PCX. When it was first developed, the limitations of the EGA card led to a palette that contained just 16 colors. PCX also supported CGA, so that the palette contained only 4 colors. Both of these palettes were stored in the palette array structure of the header. When the PCX was modified to display VGA, there was not enough room to store the palette in the header; subsequently, it was located at the end of the file.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

PGM

Full Name	PGM (Portable Graymap File Format)
Format ID	IG_FORMAT_PBM = 28 (see the Note below)
File Extension(s)	*.pgm
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

NOTE: ImageGear supports PGM file format via its `IG_FORMAT_PBM` format filter. Use `IG_FORMAT_PBM` to save grayscale images to PGM format.

ImageGear Supported versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW - Grayscale: 8, 16 bpp
- IG_COMPRESSION_ASCII - Grayscale: 8, 16 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW - Grayscale: 8, 16 bpp
- IG_COMPRESSION_ASCII - Grayscale: 8, 16 bpp

ImageGear Filter Control Parameters:

None

Comments:

This format quickly and easily transfers grayscale bitmap images. This format, as well as PBM, PNM, and PPM, are at the core of a set of utility programs also written by Jef Poskanzer. These formats serve as an intermediary storage methods for the conversion of other file formats.

The Portable Graymap File Format structure is very simple. It begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, a “maximum gray value”, and perhaps a comment line identifying the filename. The bitmap data follows white space (usually a carriage return). The number of pixels is equal to width * height. A pixel value of 0 indicates black, and a “maximum gray value” is equivalent to white.

The magic number of the header can have one of two values: either P2 or P5. P2 indicates that the bitmap data is read as ASCII decimal values. P5 indicates that the bitmap data is stored as plain bytes. This makes for a smaller and faster-to-read file.

If the maximum gray value does not exceed 255 (2^8 = 256 gray values from 0 to 255), each pixel is represented by a 8-bit sample. ImageGear loads such images as 8-bit grayscale. Otherwise, each pixel is represented by a 16-bit sample. ImageGear loads these images as 16-bit grayscale.

See Also:

PPM, PBM, PNM

References used

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest / McGraw-Hill, 1995.

PGM Specification by Jef Poskanzer, copyright © 1989, 1991.

PNG

Full Name	PNG (Portable Network Graphics)
Format ID	IG_FORMAT_PNG = 33
File Extension(s)	*.png
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Supports single alpha channel for read/write.
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- 10th draft (future drafts will be backward-compatible)
- PNG v2

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 2, 4, 8 bpp;
  - Grayscale: 1, 2, 4, 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp

ImageGear Write Support:

- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;

- Grayscale: 8, 16 bpp;
- Grayscale + Alpha: 16, 32 bpp;
- RGB: 24, 48 bpp;
- RGB + Alpha: 32, 64 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
STRIP_CONFIG	INT	IG_PNG_STRIP_FIXED_COUNT = 0	IG_PNG_STRIP_FIXED_COUNT, IG_PNG_STRIP_FIXED_BUFFER	Format of PNG strip, see IG_PNG_STRIP_... constants
BUFFER_SIZE	DWORD	0		Size of the buffer for strip if STRIP_CONFIG is IG_PNG_STRIP_FIXED_BUFFER
STRIP_COUNT	INT	0		Number of lines in one strip if STRIP_CONFIG is IG_PNG_STRIP_FIXED_COUNT
COMP_LEVEL	UINT	IG_PNG_DEFAULT_COMPRESSION	IG_PNG_DEFAULT_COMPRESSION, IG_PNG_MIN_COMPRESSION, IG_PNG_MAX_COMPRESSION	Compression level. It must be between IG_PNG_MIN_COMPRESSION and IG_PNG_MAX_COMPRESSION. Better compression takes more time for compression.

Filter Control Parameter	Type	Default Value	Available Values	Description
KEEP_ALPHA	BOOL	TRUE	TRUE, FALSE	**When loading:** if this parameter is `True`, then alpha channel is loaded into Alpha DIB. Otherwise, the image is transformed into RGB, by compositing over the background supplied in the bKGD chunk, if it is present, or over the default background. **When saving:** if this parameter is `True`, and alpha DIB is present, it will be written to the file.

Comments:

The PNG (pronounced “Ping”) format was created out of reaction to Unisys’s announcement that it would begin requiring royalty fees for use of its LZW compression scheme. This was the compression scheme for the widely-used GIF format (found in Web pages and online library images).

Thomas Boutell and a host of other programmers began working to devise a new file format to eliminate the need for payment of royalty fees. The result was a file format that offers better compression than GIF, and adds features GIF doesn't offer, including truecolor, and full alpha channel and gamma correction.

The basic organization of the PNG file format utilizes storage structures known as “chunks”. A chunk is a block of data that contains its own header (identifying the chunk size and type). This makes it easy for a PNG viewer to identify chunks and to skip over the ones that are not necessary.

The first entry in a PNG file is the “PNG signature” that identifies the format as PNG. The file then proceeds with a series of chunks.

The IHDR Image Header, or IHDR Chunk, contains a number of fields including the height, width, depth, color type, and compression type of the image. The only valid compression value is 0, which indicates the PNG’s custom compression scheme, a deflate/inflate compression with a 32k sliding window. This is a derivative of ZZ77, the precursor to LZW. ZZ77 is the compression scheme used by pkzip software.

The “PLTE Palette Chunk” contains 1 to 256 palette entries. This is present if the type field of the header chunk is set to 3. The number of colors cannot exceed the range provided by the bit depth.

The image data is stored in “IDAT Image Data Chunks”. These are subdivided into chunks whose size is usually determined by the size of the encoder’s buffer. The data may be stored in “interlaced order” allowing the image to be “faded in”. The data may also be interlaced. Interlacing helps the appearance of an image as it displays while being decompressed, so that it “fades in”. In a non-interlaced file, the presentation of the image data begins with row 1 and works downward to the last row of data. This method of display does not allow a quick preview of the whole image. When the data is interlaced, the lines are saved and displayed out of sequence. Every fourth row is displayed first and then filled in with every remaining fourth row, until all of the lines are displayed. This allows the eye to perceive the basic subject of the whole image before it is completely displayed.

“Ancillary Chunks” are optional. They must appear before the first IDAT and after the PLTE. One chunk is the “bKGD chunk” that sets the default background color for the image. Two of the other ancillary chunks are the “hHist Chunk” or histogram chunk, and the “tEXt” chunk. The histogram chunk appears if there is a palette. It stores the frequency of each color of the palette as it occurs in the image data. If the application doesn’t support all of the colors in the palette, the histogram can be used to choose a subset of colors. There may be any number of text chunks. They can vary in length from 0 to the maximum chunk size. They include the image author, copyright information, and any desired comments.

Questions about PNG can be e-mailed to: png-info@uunet.uu.net

References used:

Murray, James D. “Graphic Image Format FAQ 3-4”. James D. Murray, 1994-1996.

PNG (Portable Network Graphics), tenth draft. Page 5, copyright Thomas Boutell, May 1995.

Wegner, Tim. “Coding for PNG Graphics”, “PC Techniques”, Feb/Mar 1996, pp 32-38.

PNM

Full Name	PNM (Portable Any-Map File Format)
Format ID	IG_FORMAT_PBM = 28 (See the Note below)
File Extension(s)	*.pnm
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

NOTE: ImageGear supports PNM file format via its `IG_FORMAT_PBM` format filter. Use `IG_FORMAT_PBM` to save images to PNM format.

ImageGear Supported versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Filter Control Parameters:

None

Comments:

The PNM format is useful for quick and easy transfer of uncomplicated monochrome, gray-scale, or color bitmap images. This format, as well as the PBM, PGM, and PPM formats, are at the core of a set of utility programs also written by Jef Poskanzer. These formats serve as intermediary storage methods for the conversion of other file formats. For example, a function called gifftoppm translates a GIF file to a PPM, from where it can be then translated to a TIFF using the pnmtotiff.

The Portable Anymap File Format structure is very simple. The “Anymap” portion of its name refers to the fact that it can be one of three types of UNIX bitmap file formats: the Portable Bitmap File Format (PBM), the Portable Graymap File Format (PGM), or the Portable Pixelmap File Format (PPM). A PNM begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, a “maximum color-component value” if it is a PPM, a “maximum gray value” if it is a PGM, and perhaps a comment line identifying the filename. Following white space (usually a carriage return) is the bitmap data. For PGM and PPM files, the number of pixels is equal to width * height (whereas, in a PBM file, this calculation gives you the number of bits).

The magic number of the header can have one of six values depending on whether the bitmap is in PBM, PGM, or PPM format. Each format can have one of two magic numbers depending on whether the pixel data is stored in ASCII decimal or plain bytes. See the descriptions for the individual formats (these are all supported by ImageGear) for more details.

References used

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest / McGraw-Hill, 1995.

PBM, PGM, PPM, PNM Specifications by Jef Poskanzer, copyright © 1989, 1991.

PPM

Full Name	PPM (Portable Pixmap File Format)
Format ID	IG_FORMAT_PBM = 28 (See the Note below)
File Extension(s)	*.ppm
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

NOTE: ImageGear supports PPM file format via its `IG_FORMAT_PBM` format filter. Use `IG_FORMAT_PBM` to save truecolor images to PPM format.

ImageGear Supported versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII - RGB: 24, 48 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII - RGB: 24, 48 bpp

ImageGear Filter Control Parameters:

None

Comments:

The PPM format is useful for quick and easy transfer of color bitmap images. This format, as well as the PBM, PGM, and PNM formats, are at the core of a set of utility programs also written by Jef Poskanzer. These formats serve as intermediary storage methods for the conversion of other file formats. For example, a function called gifftoppm translates a GIF file to a PPM, where it can translate to a TIFF using the pnmtotiff.

The Portable Pixmap File Format structure is very simple. It begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, a “maximum color-component value”, and perhaps a comment line identifying the filename. Following white space (usually a carriage return) is the bitmap data. The number of pixels is equal to width * height, with each pixel being represented by three bytes: one each for Red, Green, and Blue color components, respectively.

The magic number of the header can have one of two values: either P3 or P6. P3 indicates that the bitmap data is read as ASCII decimal values. P6 indicates that the bitmap data is stored as plain bytes. This makes for a smaller and faster-to-read file.

If the maximum gray value exceeds 255 (2^8 = 256 gray values from 0 to 255), each pixel is represented by three 16-bit RGB samples, making a total of 48 bits per pixel. ImageGear loads such images to 24-bit RGB.

See Also:

PGM, PBM, PNM

References used:

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest / McGraw-Hill, 1995.

PPM Specification by Jef Poskanzer, copyright © 1989, 1991.

PTOCA

Full Name	PTOCA (Presentation Text Object Content Architecture)
Format ID	IG_FORMAT_PTOCA = 65
File Extension(s)	*.pto, *.mod
Data Type	Metafile
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, .NET, .NET64

ImageGear Supported versions:

- Format 2

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - Indexed RGB: 8 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Availabl e Values	Description
FACTOR	DOUBLE	1.0		Linear DIB size Factor.

Comments:

ImageGear can read PTOCA files as content of multi- and single-page MOD:CA files, ensuring better compatibility with IBM's AFP (Advanced Function Presentation) printer architecture.

The PT object structure starts with the "Begin Presentation Text" structure field and finishes with the "End Presentation Text" structure field. Text data is placed between these fields and consists of graphic characters and control sequences that describe how the text will be shown on the presentation surface. The PT object can include some other optional structured fields that define initial presentation text values. If PTOCA is included in the MOD:CA content, some presentation text parameters (like code page, font, an initial values) can be defined on the MOD:CA level.

References used:

IBM BookManager BookServer, Copyright 1989, 1999, IBM Corporation.

QuickTime

Full Name	QT (Quick Time movie)
Format ID	IG_FORMAT_QUICKTIMEJPEG = 76
File Extension(s)	*.jpg, *.mov, *.qt
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

NOTE: To support QuickTime format ImageGear Multimedia Component should be attached to Core ImageGear.

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_JPEG - RGB: 24 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments:

QuickTime format has been developed by Apple, inc. as a container for various media such as video and audio clips and still images.

ImageGear Core component supports loading of the first frame from JPEG- and Motion JPEG compressed QuickTime files.

RAS

Full Name	RAS (Sun Raster Data Format)
Format ID	IG_FORMAT_RAS = 37
File Extension(s)	*.ras
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_COMPRESSED	INT	0 (FALSE)	TRUE, FALSE	Compression flag. If TRUE the saving image will be RLE compressed.

Comments:

Bitmap images used under the SunOS system and UNIX imaging applications are usually stored in Sun Raster form.

The Sun Raster header contains image data (size and type), and the type and size of the colormap, if present. It also contains a Sun Raster identifying tag called “MagicNumber,” which always contains the same value. A type field identifies the version of Sun Raster, the most common are called “Old” and “Standard”. These are actually the same format and indicate that the image is not compressed. Other possible versions of Sun Raster files include TIFF and IFF, meaning that the image data was converted from one of these formats.

Following the header is a colormap, if applicable. Most 24 or 32-bit raster images do not use a colormap, but rather store the color values directly with the image data. This is known as “truecolor”.

The last element of the Sun Raster file is the image data itself. It is usually in 2D raster format.

References used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

RAW

Full Name	RAW (Raw image)
Format ID	IG_FORMAT_RAW = 58
File Extension(s)	*.raw
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Yes
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 9-16 bpp;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 9-16 bbp;

- RGB: 24 bpp;
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC:
  - Indexed RGB: 1 bpp

NOTE: To use ABIC compression scheme ImageGear ABIC Component should be attached.

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
ALIGNMENT	UINT	0	0, 1, 2, 4	Row alignment. 0: DWORD alignment 1: BYTE alignment 2: WORD alignment 4: DWORD alignment
BITS_PER_PIXEL	UINT	0		Used internally
COMPRESSION	AT_MODE	0		Used internally
FILL_ORDER	INT	0		Used internally
HEIGHT	AT_DIMENSION	0		Used internally
WIDTH	AT_DIMENSION	0		Used internally

Comments:

A raw image file contains no header or identifying information. Also, ImageGear can load images of proprietary or unsupported formats as Raw data. Since ImageGear cannot obtain parameters, such as width, height or bits per pixel from the file, the application should specify them.

Scitex CT

Full Name	SciTex (Scitex CT)
Format ID	IG_FORMAT_SCI_CT = 91
File Extension(s)	*.sct
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE - CMYK: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE - CMYK: 32 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
HEIGHT_IN_UNITS	DOUBLE			Height of the picture, in measurement units.
WIDTH_IN_UNITS	DOUBLE			Width of the picture, in measurement units.
FILE_TYPE	WORD	"CT"	"CT"	Scitex file type. The only **"CT"** - Continuous Tone Picture is currently supported.
UNITS_MEASUREMENT	BYTE		0 1	Units of measurement (0 mm, 1 inches)

Filter Control Parameter	Type	Default Value	Available Values	Description
SCAN_DIRECTION	BYTE			ScanDirection is a bitmap indicating the position of the image source on the scanner. The bitfields are defined as follows: **Bit 0** 0 = Top to bottom, 1 = Bottom to top Bit 1 0 = Left to right, 1 = Right to left Bit 2 0 = No rotation, 1 = 90 degree counter-clockwise rotation Bits 3:7 Undefined (always 0)

Comments:

ImageGear has read and write support for Scitex HandShake Continuous Tone Picture files. This is the native format used by Scitex scanners and printers for high-end image processing and color separation.

Scitex CT files store uncompressed, CMYK true-color raster data. They contain a Control Block, a Parameters Block, and the image data. Scitex CT images are typically four-color separation, CMYK, line-interleaved raster data. The separations are always stored by scan line and in the order C-M-Y-K (cyan-magenta-yellow-black). A color pixel value have up to 16 separations (128 bits) in size. Separations 1 through 4 are defined in order (C-M-Y-K). Separations 5 through 16 are reserved for future expansion of the format, as shown below.

Each row or Scitex CT image data is stored in separated color. The first separation's row data is followed by the second, and so forth, up to the number of separations specified by NumColorSeparations. Only the data for the separations defined by the SeparationsBitMask field is actually stored in the CT file. Each pixel can contain up to 16 separation components and each component is one byte in size. A CMYK pixel contains four components and is of a 32-bit size. Remember that the data is not stored by pixel, but by separation. If rows contain odd numbers of bytes, the zero padding byte will be added to the end of each separation to preserve word alignment.

Note: You can set ImageGear CMYK Support level to `IG_CONVERT_TO_RGB` to convert CMYK images to 24-bit RGB during loading. However, use of this mode is deprecated. We recommend to load the image in its native format, and then convert to desired colorspace if needed.

References used

Copyright© 1994, 1996, O'Reilly & Associates, Inc.

SGI

Full Name	SGI (Silicon Graphics Image)
Format ID	IG_FORMAT_SGI = 38
File Extension(s)	*.sgi, *.bw, *.rgb, *.rgba
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	Supports single alpha channel for read and write.
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8 bbp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp;

ImageGear Filter Control Parameters:

None

Comments:

This format was developed for use with the SGI image library included on most Silicon Graphics computers. Most SGI images are black and white.

The major components of an SGI file are the 512-byte header, a “scan-line offset table”, and the bitmap header. SGI is one of the few formats to use a scan-line offset table.

The fields of the header structure include a compression flag (1 = compressed), the height and width (in pixels) of the image, the number of bit planes, the highest pixel value and the lowest pixel value, the name of the image, and pixel format. Pixel format can indicate the number of color channels, whether the image is dithered to a single channel, and whether the bitmap image is actually a color map for other images.

The bitmap data is stored up-side-down—the first scan line is at the bottom of the bitmap. If the data is RLE, a scan-line offset table is present, following the header and preceding the bitmap. This increases compression further by allowing repeated offsets to the same scan line, if several scan lines have the same value. A gray-scale image may even refer to the same scan line from three different bit fields. For this reason, and because SGI data that is compressed can be stored in any scan-line order, the offset table must not be ignored.

References used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats, 2d ed. Sebastopol, CA: O’Reilly & Associates, Inc., 1996.

SVG

Full Name	SVG (Scalable Vector Graphics format)
Format ID	IG_FORMAT_SVG = 77
File Extension(s)	*.svg
Data Type	Vector Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64

NOTE: To support SVG format ImageGear CAD/Vector (SVG) Component should be attached to Core ImageGear.

ImageGear Supported versions:

- SVG 1.1 (partial)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - Vector data

ImageGear Write Support:

- IG_COMPRESSION_NONE - Vector data

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
EXTERN_FILE_PATH	String	""	Any string up to 262 characters	Defines a path where external files (like raster images) should be looked for. A value of "" looks in the same directory as the file being loaded.
WIDTH	Long	512	Any positive value	Defines the DIB width of the loaded image. The DIB height is calculated using the file's scale factors.

Comments:

SVG (Scalable Vector Graphics) is a platform for two-dimensional graphics. It has two parts: an XML-based file format and a programming API for graphical applications. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), raster images, and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. Text can be in any XML namespace suitable to the application, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility.

SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements from different namespaces simultaneously within the same Web page.

References used

About SVG - 2d Graphics in XML.- http://www.w3.org/Graphics/SVG/About.html

TGA

Full Name	TGA (Truevision Targa)
Format ID	IG_FORMAT_TGA = 39
File Extension(s)	*.tga, *.tpic
Data Type	Raster Image
Data Encoding	Binary

IG Multipage Support	No
IG Alpha Channel Support	Single alpha channel for read/write
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 2.0, 1991
- Version 1.0, 1984

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 1 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Premultiplied Alpha 1 bpp;
  - RGB 24 bpp + Premultiplied Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;
- IG_COMPRESSION_RLE:
  - Grayscale: 1 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Premultiplied Alpha 1 bpp;
  - RGB 24 bpp + Premultiplied Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;
- IG_COMPRESSION_RLE:
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
AUTHOR_COMMENT	STRING	" "		Author's comments.
AUTHOR_NAME	STRING	" "		Name of author.
IMAGE_ID	STRING	" "		Image ID
SOFTWARE_ID	STRING	" "		Software ID
STAMP_DAY	WORD	0		Day stamp
STAMP_HOUR	WORD	0		Hour stamp
STAMP_MINUTE	WORD	0		Minute stamp
STAMP_MONTH	WORD	0		Month stamp
STAMP_SECOND	WORD	0		Second stamp
STAMP_YEAR	WORD	0		Year stamp

Filter Control Parameter	Type	Default Value	Available Values	Description
KEEP_ALPHA	BOOL	FALSE	TRUE, FALSE	FALSE means to ignore Alpha bits. Non-zero (TRUE) means to pass alpha bits back through callback.
PROMOTE16	BOOL	FALSE	TRUE, FALSE	FALSE means to create 16 bit DIB without promoting to 24 bits. Non-zero (TRUE) means promoting.
HEADER_TYPE	INT	0	1 or 2	1 or 2 for revision level of header
STORES_TAMP	BOOL	FALSE	TRUE, FALSE	FALSE for no stamp, TRUE to save a stamp with image. HEADER_TYPE must be 2.
STAMP_WIDTH	BOOL	0	TRUE, FALSE	Stamps width must not be larger than 64 pixels
STAMP_HEIGHT	BOOL	0	TRUE, FALSE	Stamps height must not be larger than 64 pixels
PALETTE	BOOL	FALSE	TRUE, FALSE	TRUE indicates that color map should be created.
CONV_TO_16	BOOL	FALSE	TRUE, FALSE	If TRUE, convert to 16 bpp when saving.
THUMB_FLAG	UINT	0	TRUE, FALSE	TRUE if save should include a thumbnail.
THUMB_WIDTH	UINT	0		Width of thumbnail.
THUMB_HEIGHT	UINT	0		Height of thumbnail - Thumbnails are 24 bit.

Comments:

This file format was originally developed by AT&T for use with its image capture boards. The format was taken over by Truevision when it acquired the product line from AT&T. It is now commonly used for digitized images and also for high-quality images produced by ray tracers and other graphics applications.

It became a popular file format mainly because it was the first 24-bit truecolor format to come to the PC market. There are several varieties of Targa files; the most commonly used are the Targa 16, Targa 24, and Targa 32. The names are derived from the type of hardware used to create them.

The fixed-sized header information of the Targa format includes: the existence (or not) and colormap, location, size, pixel depth, image location, colormap (if it exists), and finally the image data itself.

Version 2.0 introduced a file footer that identifies it as the newest version and contains pointers to additional fields in two main structures: the “extension area” and the “developer directory”. The extension area contains the addresses of many optional fields, one of the most popular being the “postage stamp image” (miniature of the main image).

The developer directory can be used to store proprietary information. Developers can register their own private fields with Truevision. A null-terminated ASCII string containing “TRUEVISION-XFILE.”, and positioned at the end of the file, indicates that the footer is valid.

Targa defines 3 color methods: pseudo color, direct color, and truecolor. Pseudo color uses an index to a color palette. Direct color is like pseudo color except that the RGB components are looked up separately. In truecolor files, the color information is stored directly in the image data. The palettes used by Targa files are variable in size; they do not necessarily correlate to the bit depth of the image. The presence of a palette does not always mean that it is used to display the image.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

TIFF

Full Name	TIFF (Tagged Image File Format)
Format ID	IG_FORMAT_TIF = 40
File Extension(s)	*.tif, *.tiff
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	Yes
IG Alpha Channel Support	Yes
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 6.0 - Added support for CMYK and YCbCr color images, and JPEG compression. Ability to store pixels in “tiles”
- Version 5.0 - Added ability to store palette color images and support for LZW compression. This version featured TIFF “classes.”

- Version 4.0 - Added support for uncompressed RGB color images.
- Version 3.0 - First public release.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multipage file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file
- IG_FLTR_PAGESWAPSUPPORT - page swaping in multi-page files
- IG_FLTR_MPDATASUPPORT - faster multi-page access by storing private format data (used only with IG_mpi_... and IG_mpf_... API)

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
  - RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
  - Indexed RGB + Extra: 1, 2, 4, 8 bpc;
  - Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;

- Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
- RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
- Indexed RGB + Extra: 1, 2, 4, 8 bpc;
- Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
- CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
- Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
- RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.

- IG_COMPRESSION_HUFFMAN:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
  - RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
  - Indexed RGB + Extra: 1, 2, 4, 8 bpc;
  - Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;

- CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
- Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
- RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.

- IG_COMPRESSION_LZW:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
  - RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
  - Indexed RGB + Extra: 1, 2, 4, 8 bpc;
  - Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.

NOTE: To use LZW (Lempel-Ziv-Welch) compression scheme ImageGear GIF/TIFF-LZW Component should be attached.

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 4, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
  - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;

        - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
    - IG_COMPRESSION_PACKED_BITS:
        - Indexed RGB: 1, 4, 8 bpp;
        - Grayscale: 8, 12, 16, 32 bpp;
        - RGB: 24, 36, 48 bpp;
        - Lab: 24, 36, 48 bpp;
        - CMYK: 4, 32, 48, 64 bpp;
        - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
        - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
        - Lab + Extra: 8, 12, 16 bpc;
        - CMYK + Extra: 8, 12, 16 bpc;
        - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
        - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
    - IG_COMPRESSION_HUFFMAN:
        - Indexed RGB: 1 bpp;
    - IG_COMPRESSION_CCITT_G3:
        - Indexed RGB: 1 bpp;
    - IG_COMPRESSION_CCITT_G4:
        - Indexed RGB: 1 bpp;
    - IG_COMPRESSION_CCITT_G32D:
        - Indexed RGB: 1 bpp;
    - IG_COMPRESSION_JPEG (Lossy, Progressive):
        - Grayscale: 8, 12 bpp;
        - RGB: 24, 36 bpp;
        - Lab: 24, 36 bpp;
        - CMYK: 32, 48 bpp;
    - IG_COMPRESSION_JPEG (Lossless):
        - Grayscale: 8, 16 bpp;
        - RGB: 24 bpp;
    - IG_COMPRESSION_DEFLATE:
        - Indexed RGB: 1, 4, 8 bpp;
        - Grayscale: 8, 12, 16, 32 bpp;

- RGB: 24, 36, 48 bpp;
- Lab: 24, 36, 48 bpp;
- CMYK: 4, 32, 48, 64 bpp;
- Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
- RGB + Premultiplied Alpha: 32, 48, 64 bpp;
- Lab + Extra: 8, 12, 16 bpc;
- CMYK + Extra: 8, 12, 16 bpc;
- Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
- RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;

- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 4, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
  - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;

NOTE: To use LZW (Lempel-Ziv-Welch) compression scheme ImageGear GIF/TIFF-LZW Component should be attached.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
STITCH_TILES	BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.

Filter Control Parameter	Type	Default Value	Available Values	Description
MISSING_COMPRESSION	Long	0		Missing compression
READ_JPEG_AS_YCBCR	BOOL	FALSE	TRUE, FALSE	For internal use.
UPDATE_LUT16	BOOL	TRUE	TRUE, FALSE	
UPDATE_PAGE_NUMBERS	BOOL	TRUE	TRUE, FALSE	
BUFFER_SIZE	DWORD	32768	any DWORD>0	This parameter specifies the buffer size for each strip for write operation if WRITE_CONFIG==IG_TIF_STRIP_FIXED_BUFFER
FILL_ORDER	MODE	IG_FILL_MSB	IG_FILL_MSB IG_FILL_LSB	Specifies fill order (tag 266) for tiff file to be written
BIG_ENDIAN	BOOL	FALSE	FALSE, TRUE	If TRUE, big endian order is used for write operation (“MM”), in other case little endian is used(“II”).
DOCUMENT_NAME	LPCHAR	""	any string	Specifies value for DocumentName (tag 269) to write into image.
DATETIME	LPCHAR	""		Specifies value for DateTime (tag 306) to write into image.
IMAGE_BEFORE_IFD	BOOL	FALSE	FALSE, TRUE	This flag specifies physical location of raster data inside TIFF file relatively to IFD record. If this value is TRUE then image data is to be written before IFD record.

Filter Control Parameter	Type	Default Value	Available Values	Description
WRITE_CONFIG	MODE	IG_TIF_STRIP_FIXED_COUNT	IG_TIF_STRIP_FIXED_COUNT, IG_TIF_STRIP_FIXED_BUFFER, IG_TIF_TILED_FIXED_SIZE, IG_TIF_TILED_FIXED_COUNT	Specifies configuration of TIFF file to be written: IG_TIF_STRIP_FIXED_COUNT - writes fixed number of strips. IG_TIF_STRIP_FIXED_BUFFER - writes strips of size no more then given size IG_TIF_TILED_FIXED_SIZE - writes image in tiles of specified size IG_TIF_TILED_FIXED_COUNT - devides image into specified number of tiles vertically and horizontally.
PLANAR	BOOL	FALSE	FALSE, TRUE	Specifies tag value 284 for output image.
NUMBER_OF_STRIPS	UINT	1	any value >0	If WRITE_CONFIG==IG_TIF_STRIP_FIXED_COUNT then this value is used as the number of strips to be written
WRITE_CLASS_F	BOOL	FALSE	FALSE, TRUE	If this value is TRUE then image to be written in TIFF format compatible with class F requirements.
NEW_SUBFILE_TYPE	UINT	0xFF		This parameter specifies value for tag 254. If value of this control parameter 0xFF then default value is used according to TIFF 6.0 format specification.
INCLUDE_PAGE_NUMBER	BOOL	TRUE	FALSE, TRUE	If this parameter is TRUE then include tag 297 into TIFF image with real value of page number.
WRITE70	BOOL	TRUE	FALSE, TRUE	If this value is TRUE then output TIFF-JPEG image will be produced in TIFF 7.0 compatible format but in other case it will be compatible with TIFF 6.0.
DO_NOT_WRITE_PALETTE	BOOL	FALSE	FALSE, TRUE	Set to **TRUE** to skip the palette when writing a TIFF.

Filter Control Parameter	Type	Default Value	Available Values	Description
PHOTOMETRIC	UINT	IG_TIF_PHOTO_WHITEZERO	IG_TIF_PHOTO_WHITEZERO, IG_TIF_PHOTO_BLACKZERO, IG_TIF_PHOTO_RGB, IG_TIF_PHOTO_CMYK, IG_TIF_PHOTO_PALETTE, IG_TIF_PHOTO_TRANSPARENCY, IG_TIF_PHOTO_YCBCR, IG_TIF_PHOTO_CIELAB	Specifies photometric interpretation (tag 262) for write operation.
TILE_H_COUNT	DWORD	10		If WRITE_CONFIG==IG_TIF_TILED_FIXED_COUNT then this value is used as number of tiles in horizontal dimension.
TILE_V_COUNT	DWORD	10		If WRITE_CONFIG==IG_TIF_TILED_FIXED_COUNT then this value is used as number of tiles in vertical dimension.
TILE_WIDTH	DWORD	64		If WRITE_CONFIG==IG_TIF_TILED_FIXED_SIZE then this value is used to specify horizontal dimension of each tile.

Filter Control Parameter	Type	Default Value	Available Values	Description
TILE_HEIGHT	DWORD			If WRITE_CONFIG==IG_TIF_TILED_FIXED_SIZE then this value is used to specify vertical dimension of each tile.
IMAGE_HEIGHT	DWORD	0	Any positive value	Was used internally in previous versions of IG.
IMAGE_WIDTH	DWORD	0	Any positive value	Was used internally in previous versions of IG.
SAVE_DIFF_PREDICTOR	BOOL	FALSE	FALSE, TRUE	If this parameter is TRUE, then output TIFF-LZW image will be produced using the horizontal differencing predictor.
SUBIFD_PATH	LPCHAR	Empty string		Number of the SubIFD to load the image from. If set to empty string (default), load image from root IFD. See “Camera Raw Image support” section for more detail.
FAST_PAGE_COUNT	BOOL	FALSE	FALSE, TRUE	Affects page counting. When FALSE, ImageGear counts only those IFDs that contain images. When TRUE, ImageGear counts all IFDs, without checking for presence of images in them. The latter mode requires significantly less reading operations and thus works faster, especially if the image is accessed through a network.

Comments:

TIFF was developed for use in storing black-and-white images from scanners and desktop publishing applications. Now, in its fourth release (version 6.0), it is one of the most detailed and versatile bitmap formats in use. It is supported by most art, imaging, and word-processing applications. It supports several compression schemes. Aside from saving image data in bitmap form, it can also contain vector or text-based images.

Containing just three fields, the TIFF header is simple and one of the shortest of all the graphics file format headers. But, the structure of a TIFF is complicated, with variable length fields, variable number of fields, and the ability to store information (other than the header) in any order desired.

The other two major components of the TIFF format are “Image File Directories” (IFDs) and the image or images themselves. There is one IFD per image stored. The combination of an IFD and an image is referred to as a “subfile”. The header contains an offset pointer to the first IFD. If there are multiple IFDs, each contains an offset to the next. The last IFD contains a value that signifies the end of the file.

IFDs closely resemble a header structure, and the information stored in them is often referred to as “TIFF Header Information”. Unlike a header, however, they contain a variable number of “tags” (pointers or fields). In addition, each tag can point to data with a variable length. TIFFs are notorious for the number of tags that they can contain, up to a maximum of 65,535 tags of nearly 100 different types (version 6.0). Tags are listed in order by code number so that a TIFF reader can easily determine what fields are present. While ImageGear reads and stores all TIFF tags, it utilizes a subset of all of the possible tags. See note on previous page.

In version 5.0, the presence of certain subgroups of tags determined what “class” the TIFF belonged to. The classes are: TIFF-B–monochrome, TIFF-F–fascimile, TIFF-G–gray scale, TIFF-P–palette based, TIFF-R–RGB color, TIFF-X–any class, TIFF-Y–can use JPEG compression. Version 6.0 uses tags to divide the TIFF type into different file configurations, leaving behind the class concept. Version 6.0 configurations are: Bilevel, palette color, RGB, gray scale, YCbCr, and Class F (fascimile).

TIFF bitmap data can be stored in one of two configurations: strips or tiles. Strips are groups of adjoining rows of bitmap data, and can be found in version 5.0 and 6.0 files. Tiles were new to TIFF version 6.0. They are rectangular or square sections of bitmap data. The method of storage is determined in part by what kind of compression (if any) is used. JPEG compression can handle tiled images. Due to the need for padding with tile storage, tiling is not usually efficient for small images. For more information, see the section entitled Working with TIFF Files.

ImageGear supports the following compression schemes for TIFF:

- Uncompressed
- CCITT Group 3
- CCITT Group 3 2D
- CCITT Group 4
- Huffman
- JPEG
- Lossless JPEG
- LZW (Lempel-Ziv-Welch)
- Packbits
- Progressive JPEG
- Deflate

See the ImageGear Supported Compressions Reference for descriptions of these compression types. The compression tag of the IFD tells whether the image is compressed, and by what method. (Not all TIFF files can use JPEG compression. It is supported by version 6.0, but in version 5.0, only a “Y” class TIFF can use JPEG). (Only 8-bit TIFF files can use LZW compression).

TIFF/EP

TIFF/EP format was designed to provide a means for storing “raw” (unprocessed) image from digital camera’s sensor.

TIFF/EP allows to store several versions of the same image in one file. Typically, TIFF/EP image includes a small preview and a raw image. It can also include a larger or full-size preview, or some other variations of the image.

TIFF/EP uses IFD trees for storing different versions of image. This is different from IFD chains that are used in baseline TIFF to store multiple pages.

ImageGear does not detect TIFF/EP as a separate file format. If Raw detection is enabled, and ImageGear can recognize TIFF/EP file as a known Raw format, it detects and loads the file as DCRAW. Otherwise it detects TIFF/EP as TIFF. One of well known extensions to TIFF/EP is Adobe DNG format. ImageGear detects it as a separate file format.

Most of digital cameras store pixels in "mosaic" format. At a given pixel location either a Red, Green, Blue, Cyan, or some other color sample value is recorded. Such images are referred to as "Color Filter Array" type. TIFF/EP format uses a two-dimensional matrix called "Color Filter Array pattern" to describe positions of pixels of particular color in the mosaic image. In order to recreate the full color values in each pixel, it is necessary to interpolate intensities of neighbouring pixels.

To enable loading images from TIFF SubIFDs, a new control parameter is added to TIFF filter: "SUBIFD_PATH". This parameter has the type of String, and its default value is "" (empty string). By default, ImageGear loads the image in the root IFD (thumbnail). If "SUBIFD_PATH" string begins with a number, then ImageGear loads the image from the corresponding SubIFD of the root IFD. For example, if SUBIFD_PATH is set to "3", ImageGear will load the image from the 3rd SubIFD.

SUBIFD_PATH parameter also affects metadata reading. Metadata is loaded starting from the IFD specified by SUBIFD_PATH.

NOTE: ImageGear does not support color reconstruction of TIFF/EP images, unless they are detected to be a Adobe DNG or a DCRAW image supported by ImageGear.

References used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. “Graphic Image Format FAQ 3-4”. James D. Murray, 1994-1996.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

TXT (ASCII Text)

Full Name	TXT (ASCII Text)
Format ID	IG_FORMAT_TXT = 41
File Extension(s)	*.txt
Data Type	Raster Image
Data Encoding	ASCII
IG Multipage Support	Yes
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_ASCII - Indexed RGB: 1 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
XDPI	UINT	200		Horizontal resolution of image.
YDPI	UINT	200		Vertical resolution of image
MARGINS_LEFT	LONG	1000		Left text margin on page, expressed as thousandths of an inch.

Filter Control Parameter	Type	Default Value	Available Values	Description
MARGINS_TOP	LONG	1000 (750 - for Unix)		Top text margin on page, expressed as thousandths of an inch.
MARGINS_RIGHT	LONG	1000		Right text margin on page, expressed as thousandths of an inch.
MARGINS_BOTTOM	LONG	1000 (750 - for Unix)		Bottom text margin on page, expressed as thousandths of an inch.
PAGE_WIDTH	DIMENSION	8500		Width of resulting page, expressed as thousandths of an inch.
PAGE_HEIGHT	DIMENSION	11000		Height of resulting page, expressed as thousandths of an inch
POINT_SIZE	INT	10 (-1 - for Unix)		Font metric: If 0 then lines per page and character per line is used, else - specify font size.
WEIGHT	UINT	FALSE (0)	TRUE, FALSE	Font metric: if TRUE use bold font.
ITALIC	BOOL	FALSE	TRUE, FALSE	Font metric: if TRUE use italic font.
TAB_STOP	UINT	3 (4 - for Unix)		The number of characters per tab.
TYPE_FACE	LP | CHAR	"\x00" ("courier" - for Unix)		Font metric:typeface name of the font, If empty string then default font used “Courier new”.
LINES_PER_PAGE	UINT	0 (60 - for Unix)		Number of line per page.
CHAR_PER_LINE	UINT	0 (80 - for Unix)		Number of characters per line.
COMPATIBILITY_MODE	BOOL	FALSE (TRUE - for Unix)	TRUE, FALSE	If TRUE use old algorithm, otherwise use ImageGear Algorithm

Comments:

This is a widely used format for storing plain text files. ASCII data can also be used to give vector data instructions, but this is not supported by ImageGear.

The current, commonly used version of ASCII uses a 7-bit format and is known as “Full” or “Extended ASCII”. The 128 (2^7) different data values include printable and non-printable values. The non-printable values are represented by the first 32 (0-31) values of ASCII, and are called “control values”. They are used to communicate with screens or printers for placement of the characters. These control values represent tabs, line feeds, spaces, etc. Combinations of these values create “escape sequences” whose values are device-dependent upon implementation. To keep an ASCII file completely device-independent, a file usually does not contain any control values other than tab, line feed, and carriage return.

What makes a TXT file different from many bitmap formats is the byte order. A file is written in the natural order that it appears when output. There is no division into bit planes, or reverse order of bits and bytes. The eighth bit of each byte is normally set to zero. In older versions of TXT files, this was often used as a parity bit.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest / McGraw-Hill, 1995.

WBMP

Full Name	WBMP (Wireless Bit-Map)
Format ID	IG_FORMAT_WBMP = 66
File Extension(s)	*.wbmp,
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 1.1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

No

Comments

The WBMP (Wireless Bit-Map) read/write format is optimized to support mobile computing devices that use the Wireless Application Protocol (WAP).

File contains small header with image parameters and array or pixels in uncompressed form.

References used:

WAP WAE Specification Version, 24 May 1999.

WMF

Full Name	WMF (Windows Metafile Format)
Format ID	IG_FORMAT_WMF = 44
File Extension(s)	*.wmf,
Data Type	Metafile Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix (read raster portion only). MAC (read raster portion only)

ImageGear Supported versions:

- Version 1 Metafiles prior to Windows 3.0
- Version 2 Metafiles for Windows 3.0 and later

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
UPSIDE_DOWN	BOOL		TRUE, FALSE	If TRUE then images will be saved upside-down
TYPE	UINT		BMP_TYPE_BMC, BMP_TYPE_BMI, BMP_TYPE_BMI2	Type of the BMP file to be saved as part of WMF.
COMPRESSION	DWORD		BMP_COMP_RLE4, BMP_COMP_RLE8, BMP_COMP_RGB	Type of compression of BMP file.
TRUE_METAFILE	BOOL	TRUE (for Windows) FALSE(otherwise)	TRUE, FALSE	TRUE means executing of metafile commands (playing of metafile). Can be TRUE only for Windows.
RESOLUTION_X	DWORD	NULL		X resolution. 0 for actual resolution

Filter Control Parameter	Type	Default Value	Available Values	Description
RESOLUTION_Y	DWORD	NULL		Y resolution. 0 for actual resolution
DEPTH	DWORD	NULL		Bit Depth. 0 for actual depth

NOTE: If the TRUE_METAFILE parameter tis set to TRUE, the image will be opened as an RGB DIB for use with the GDI functions, which produce the image output. This causes the image to look like it is 24-bit per pixel, and 1024x1024 in dimensions. If the TRUE_METAFILE parameter is set to FALSE, then ImageGear will open the image according to its correct bit depth and dimensions.

Comments

A Microsoft Windows Metafile holds vector and bitmap graphics data in memory or on disk. Although it was developed for use with Windows applications, it is now used by many non-Windows-based applications, allowing data to be transferred to and from Windows applications. Due to the great success of the Microsoft Windows interface, the Windows Metafile format is found in nearly all graphical applications. Metafiles use much less space and are more device-independent than bitmaps.

See also the section Support for Metafile Formats.

The Windows metafile begins with a short header and is followed by one or more records of data. The header describes the record data. A “placement” header can also be added before the file header; it contains information needed to move the metafile between applications. Each record corresponds to a binary-encoded Windows graphics device interface (GDI) call, and contains the size of the record, the unique function number for the GDI and an array of parameters. The GDI is used by Windows to perform all image output. When the metafile is “played”, (this Microsoft term is a companion term to the Windows function named “PlayMetaFile”), each record makes a call to the appropriate function call for displaying each object in the image. The last record in the file contains a function number of zero to indicate that the end of the record data has been reached.

References used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

Petzold, Charles. Programming Windows: The Microsoft Guide to writing applications for Windows 3. Redmond, WA: Microsoft Press, 1990.

WPG

Full Name	WPG (WordPerfect Graphics Metafile)
Format ID	IG_FORMAT_WPG = 42
File Extension(s)	*.wpg
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- WPG for WP 5.1 and up can store bitmap and vector graphics in the same file.
- WPG for WP 5.0 and prior can store only bitmap or vector graphic, but not both in same file.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_RLE - Indexed RGB: 1, 4, 8 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

No

Comments

This format was created specifically for use with WordPerfect software products. WPG files for WordPerfect versions 5.1 and up can store both bitmap and vector image data in the same file.

The WordPerfect Graphics Metafile contains a short header or “prefix” (as WordPerfect Corporation referred to it). The header is followed by a record area, that is a sequence of objects and their attributes. The first record is called the “Start WPG Data” record and contains information on the size of the images and the version number of the .WPG file. The next record is usually a color map, unless the image is black and white. The next record is a bitmap record. If there

are multiple images, there is a bitmap record for each image. The last record in a .WPG file contains a NULL body to signify the end of the file. These files may also contain Encapsulated PostScript (EPS) data.

References used

Murray, James D. “Graphic Image Format FAQ 3-4”. James D. Murray, 1994-1996.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

XBM

Full Name	XBM (X BitMap)
Format ID	IG_FORMAT_XBM = 43
File Extension(s)	*.xbm
Data Type	Raster Image
Data Encoding	ASCII
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 11, data stored as 1-byte character, 1986
- Version 10, data stored as 2-byte “short” integers

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_ASCII - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_ASCII - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

No

Comments

The XBM format is intended as a convenient storage method for small monochrome images, for example, cursor and icon bitmaps. It can, however, support images of any size, but since it supports no native compression scheme, an exterior compression program is used when compacting is desired. The bitmap data is stored as ASCII data with C language syntax, making the file easy to insert into C program code. XBM data can be stored as a standalone graphics file, or within a C program header file. See also XPM, XWD.

XBM files begin with two to four #define statements in substitution of a header. These identify the image width and height, and optionally, the coordinates of a Hotspot, if one exists.

The image data follows and is more free-form than the other bitmap data formats described in this chapter. It consists of one variable-length static array of pixel values. Each value (in version 11) consists of one byte of data, and therefore represents 8 1-bit pixels. The first pixel (0,0) is represented by the high bit of the first byte in the array. Due to this one-array format, there is nothing in the data that explicitly marks the rows of the bitmap.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

XPM

Full Name	XPM (X PixMap)
Format ID	IG_FORMAT_XPM = 45
File Extension(s)	*.xpm
Data Type	Raster Image
Data Encoding	ASCII (in C language syntax)
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 3.2g, April 1993

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp

ImageGear Filter Control Parameters:

No

Comments

The XPM format was created as an extended version of the XBM file format. It is the informal standard for storing X Window pixmap data, including Hotspot information for cursor bitmaps. The image data is stored in ASCII text characters that are formatted as a standard C array of character strings. It is intended to be human-readable, is readily inserted into C/C++ program code, and can contain any number of comment lines. It therefore does not support a native compression scheme. If compacting is desired, an external compression program may be used. See also XBM, XWD.

All XPM files begin with a C language comment line containing “XPM”. Following this are three sections of data: values, colors, and pixels, and an optional fourth section: extensions. The “values” section is the equivalent of the header structure typically found in a graphics file. It gives the size of the pixmap, as well as its number of colors, characters per pixel, the location of the Hotspot (if any), and an indicator of whether the file contains an extension section. Each section is set off with a comment-line title.

The colors section contains codes for the pixmap data characters. All pixels that make up the pixmap are assigned to one or more ASCII characters and one or more colors. (e.g. the character “X” may be assigned to the color red). There are several different conventions for identifying a color. If the string “None” appears as the color to be applied to a specific character, the character(s) symbolizes a transparent pixel.

The “pixels” section contains the bitmap data that appear as an array of character strings, where one row of bitmap data is represented by one array element. Each row is a group of characters set off by quotation marks. Each character is defined in the previous “colors” section.

If indicated by the values section, an extension section appears. It can contain one or more subsections that conform to one of two syntactical formats. An “XPMENDEXT” marker is always used to mark the end of the extension section.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O’Reilly & Associates, Inc., 1994.

XWD

Full Name	XWD (X Window Dump)
Format ID	IG_FORMAT_XWD = 47
File Extension(s)	*.xwd, *.wd (for Unix)
Data Type	Raster Image
Data Encoding	Binary
IG Multipage Support	No
IG Alpha Channel Support	No
IG Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported versions:

- Version 7 for X11, June 1987 (X10 gray scale and palette only)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;

- RGB: 24 bpp

ImageGear Filter Control Parameters:

No

Comments

An XWD file can contain a representation of the window, the background, or the entire screen display. It has been designed to be a very versatile, device-independent format. See also XBM, XPM.

The general structure of the XWD graphics format begins with a long header, that is sometimes followed by a palette and contains the bitmap data. The header contains integer data and stores the header size, the XWD version, the size and location of the bitmap, the window size, location, and border width. A ByteOrder field indicates whether the bytes are stored in big-endian or little-endian order.

In the interest of making this format device-independent, the XWD supports six “visual classes” and three image formats. The visual class code is stored in the visual_class field of the header, and represents the following categories:

- Static Gray, for most monochrome screens and using a fixed device-dependent color map;
- GrayScale, for monochrome screens and using a software-supplied palette;
- StaticColor, which uses a fixed device-dependent palette;
- Pseudocolor, which uses a software-supplied palette and is intended for VGA screens;
- TrueColor, with fixed device-dependent mapping of RGB values to screen colors;
- DirectColor, with software-supplied mapping of RGB values to screen values.

The image-format categories, whose codes are stored in the pixmap_format field of the header, are called XYBitmap (1-bit), XYPixmap (single plane), and ZPixmap (two or more planes).

Where the value of pixmap_format indicates GrayScale, PseudoColor or DirectColor, a palette follows the header.

The image data is the last structure in the file. The bytes are stored in rows with groupings called “units”, whose lengths are determined by the bitmap_unit field of the header.

If the pixmap_format field is 1, indicating an XYPixmap, there are multiple representations of the bitmap data, one for each color plane, where the first bitmap represents the highest bit of the data; the second bitmap represents the second-highest bit, and so on. An image with a bit depth of 4 yields a file with four bitmaps.

References used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

ImageGear Alpha Channel Support

ImageGear provides alpha channel support for the following file format filters:

Format	Support
Adobe PSD	Supports single Alpha channel for Read only. Supports aditional Alpha channels as ImageGear Extra channels for Read only.
CUR	Supports single 8-bit alpha channel for read/write. Supports single 1-bit alpha channel for read only.
EXIF-JPEG	Supports single Alpha channel for Read only.
EXIF-TIFF	Supports single Alpha channel for read only. Supports additional Alpha channels as ImageGear Extra channels for read only.The following compressions are supported with alpha channel: ■ Uncompressed ■ Packed Bits ■ LZW ■ Deflate
FlashPix	Supports single Alpha channel for read only. Converts RGB + Alpha to RGB during loading.
ICO	Supports single 8-bit alpha channel for read/write. Supports single 1-bit alpha channel for read only.
JPEG	Supports single Alpha channel for Read only.
JPEG2000	Supports single Alpha channel for read/write, for Grayscale and RGB images.
JPX	Supports single Alpha channel for read/write, for Grayscale and RGB images. Supports additional Alpha channels as ImageGear Extra channels for read/write.
MAC PICT	Supports single alpha channel for read only. Alpha channel have to be 8-bit image.
Motion JPEG2000	Supports single Alpha channel for read only, for Grayscale and RGB images.
PNG	Suports single Alpha channel for read/write.

Format	Support
RAW	Supports single Alpha channel in uncompressed images for read only.
SGI	Supports single alpha channel for read and write.
TGA	Supports single alpha channel for read and write.
TIFF	Supports single Alpha channel for read and write. Supports additional Alpha channels as ImageGear Extra channels for read and write.The following compressions are supported with alpha channel: ■ Uncompressed ■ Packed Bits ■ LZW ■ Deflate

ImageGear Supported Digital Camera RAW Formats

Camera manufacturers do not publish specifications on their raw formats. ImageGear attempts to reconstruct color images correctly where it is possible. For some cameras, however, not all of the color balance data is available. The application may need to apply additional processing to such images. This table lists the level of support that ImageGear provides for particular cameras.

The table uses the following notation:

- S1 - Not supported in this version
- S2 - ImageGear loads non-demosaiced image, basic metadata, and exif metadata if available. Reconstructed image can have incorrect colors.
- S3 - Same as S2 plus correct white balance, black level tags. Display can be slightly different from native viewer, due to lack of info on camera colorspace.
- S4 - Displays with colors very close to native viewer
- S5 - Displays with colors and contrast very close to native viewer

Manufacturer	Model	Level of Support
Canon	EOS-10D	S5
Canon	EOS-1D	S2
Canon	EOS-1D Mark II	S5
Canon	EOS-1D Mark II N	S1
Canon	EOS-1Ds	S2
Canon	EOS-1Ds Mark II	S1

Manufacturer	Model	Level of Support
Canon	EOS-20D	S3
Canon	EOS-20Da	S1
Canon	EOS-300D	S5
Canon	EOS-30D	S4
Canon	EOS-350D	S5
Canon	EOS-400D	S4
Canon	EOS-5D	S4
Canon	EOS-D30	S2
Canon	EOS-D60	S5
Canon	EOS Digital Rebel	S5
Canon	EOS Digital Rebel XT	S5
Canon	EOS Digital Rebel XTi	S4
Canon	EOS Kiss Digital	S5
Canon	EOS Kiss N Digital	S5
Canon	EOS Kiss X Digital	S4
Canon	PowerShot 600	S2
Canon	PowerShot A5	S2
Canon	PowerShot A50	S2
Canon	PowerShot G1	S5
Canon	PowerShot G2	S5
Canon	PowerShot G3	S3
Canon	PowerShot G5	S2
Canon	PowerShot G6	S2
Canon	PowerShot Pro1	S5
Canon	PowerShot Pro70	S2
Canon	PowerShot Pro90 IS	S2
Canon	PowerShot S30	S2
Canon	PowerShot S40	S5
Canon	PowerShot S45	S2
Canon	PowerShot S50	S4

Manufacturer	Model	Level of Support
Canon	PowerShot S60	S2
Canon	PowerShot S70	S2
Contax	N Digital	S1
Epson	R-D1	S1
Fujifilm	FinePix F700	S2
Fujifilm	FinePix S2 Pro	S2
Fujifilm	FinePix S20 Pro	S2
Fujifilm	FinePix S3 Pro	S5
Fujifilm	FinePix S5000 Z	S2
Fujifilm	FinePix S5100 Z	S1
Fujifilm	FinePix S5500 Z	S1
Fujifilm	FinePix S7000 Z	S2
Fujifilm	FinePix E550	S5
Imacon	Ixpress	S1
Kodak	DCS 720x	S2
Kodak	DCS 720x (YUV)	S1
Kodak	DCS 760 C	S4
Kodak	DCS 760 M (monochrome)	S2
Kodak	DCS PRO 14n (13.5 MegaPixel)	S2
Kodak	DCS PRO 14n (3.2 MegaPixel)	S1
Kodak	DCS PRO 14n (6 MegaPixel)	S1
Kodak	DCS Pro 14n (YUV)	S2
Kodak	DCS Pro 14nx	S2
Kodak	DCS Pro SLR/c	S2
Kodak	DCS Pro SLR/c (YUV)	S2
Kodak	DCS Pro SLR/n (13.5 MegaPixel)	S2

Manufacturer	Model	Level of Support
Kodak	DCS Pro SLR/n (3.2 MegaPixel)	S1
Kodak	DCS Pro SLR/n (6 MegaPixel)	S1
Kodak	DCS Pro SLR/n (YUV)	S2
Kodak	EasyShare P850	S2
Kodak	EasyShare P880	S2
Konica Minolta	Alpha Sweet	S5
Konica Minolta	DiMAGE 5	S5
Konica Minolta	DiMAGE 7	S5
Konica Minolta	DiMAGE 7Hi	S5
Konica Minolta	DiMAGE 7i	S4
Konica Minolta	DiMAGE A1	S5
Konica Minolta	DiMAGE A2	S5
Konica Minolta	DYNAX/MAXXUM 5D	S5
Konica Minolta	DYNAX/MAXXUM/ ALPHA 7D	S5
Leaf	Valeo 11	S5
Leaf	Valeo 22	S5
Leaf	Valeo 6	S5
Leica	Digilux 2	S5
Nikon	Coolpix 5000	S2
Nikon	Coolpix 5400	S5
Nikon	Coolpix 5700	S5
Nikon	Coolpix 8400	S1
Nikon	Coolpix 8700	S5
Nikon	Coolpix 8800	S5
Nikon	D1	S5
Nikon	D100	S5
Nikon	D1H	S5
Nikon	D1X	S5

Manufacturer	Model	Level of Support
Nikon	D200	S5
Nikon	D2H	S5
Nikon	D2X	S1
Nikon	D50	S5
Nikon	D70	S5
Nikon	D70s	S5
Nikon	D80	S4
Olympus	C-5050 Zoom	S5
Olympus	C-5060 Zoom	S5
Olympus	C-8080	S5
Olympus	E-1	S5
Olympus	E-10	S5
Olympus	E-20	S5
Olympus	E-300	S1
Olympus	E-500	S5
Panasonic	DMC-LC1	S5
Panasonic	DMC-FZ30	S1
Pentax	*istD	S5
Pentax	*istDL2	S4
Pentax	*istDS	S5
Pentax	K100D	S4
Pentax	K110D	S4
Sigma	SD10	S1
Sigma	SD9	S1
Sony	DSC-F828	S2
Sony	DSC-V3	S2
Sony	DSC-R1	S3
Sony	DSLR-A100	S5

AccuSoft Image Guarantee

ImageGear is guaranteed to read all valid raster images for supported formats. If for some reason, you cannot read an image with our library, three things may have occurred:

1. The image is corrupted.
2. The image is improperly coded.
3. ImageGear is not reading this image properly.

See Also:

Identifying the Problem.

Identifying the Problem

In the rare instance that ImageGear cannot read an image, please try the following:

1. Try to read the image with another software package. Be sure to use a different package from the one that created the image. Some products that claim to produce a particular graphics file format do not adhere completely to the format specifications. If you can successfully read the image with a different product, you can feel confident that it is a valid image.
2. If you can read the file with a package other than the one that created the image, please refer to the section Sending the Image to Accusoft for the instructions how to send us your image.

Check the error count that is returned by the ImageGear function (`AT_ERRCOUNT`). If it is greater than 0, use the **`IG_error_get()`** function to get the ImageGear error codes. (`AT_ERRCODE`). For more information on doing this, you can consult either Core Component API Function Reference Chapter or Detecting and Handling Errors.

NOTE: ImageGear contains filters that are designed to decompress an image even if it is corrupted. Some images may be too badly corrupted for the filters to use. If this is the case, you will receive the error message `IGE_CORRUPTED_FILE` defined in **`GEARERRS.H`**.

Sending the Image to Accusoft

To send your image to AccuSoft, please log a support ticket via our web-ticketing system, and attach the file once the ticket is created, so that our Customer Support Team can address your problem as soon as possible.

See Also:

AccuSoft Customer Support

Software Support and Maintenance Policy

Appendix A: Function Error Return Codes

This Appendix represents ImageGear error codes including their numerical numbers and short descriptions. All error codes are listed in numerical order and divided on the groups that are specific for ImageGear components functionality. Please refere to `geararrs.h` and `gearestr.h` files for additional information about ImageGear error and warning codes.

ART Component Error Codes

`IGE_ART_WANG_INCOMPATIBLE_FILE`	-9500	Internal format of loaded annotation file is not compatible with Wang.

NRA Error Codes

`IGE_INVALID_PARAMETER_FOR_PROCESSING_WITH_REGION`	-9001	Operation can not be done using region.
`IGE_INVALID_REGION_DATA`	-9000	Invalid region

OS2 Error Codes

`IGE_UNREGISTERED_HAB`	-8000	HAB is not registered

VBX/OCX Level Error Codes

`IGE_GUIWINDOW_TYPE_INVALID`	-7020	GUIWindow type invalid
`IGE_ROTATE_ENUMERATED_VALUES_NOT_USED`	-7010	The enumerated values for rotate were not used
`IGE_VBX_INVALID_FUNCTION_NUM`	-7000	Invalid VBX function number

IGE_GUI_GET_DEVICE_RECT_FAILED	-6522	Failed to retrieve device rectangle from SourceView's page display.
IGE_GUI_DESTINATION_VIEW_UNAVAILABLE	-6521	DestinationView property must be setup in stationary mode before starting mouse tracking.
IGE_GUI_SOURCE_VIEW_UNAVAILABLE	-6520	SourceView property must be setup before starting mouse tracking.
IGE_GUI_WINDOWING_FAILED	-6519	Failed to attach to the source view control window.
IGE_GUI_SLIDER_SETUP_FAILED	-6518	Failed to setup magnifier's on-source sliding area.
IGE_GUI_INTERNAL_ERROR	-6517	Internal GUI error.
IGE_GUI_UNKNOWN_FILE_FORMAT	-6516	Unknown file format
IGE_GUI_GET_HWND	-6515	Could not retrieve the underlying HWND for this object.
IGE_GUI_GET_DLG_CTL_FAILED	-6514	Could not retrieve the dialog control object.
IGE_GUI_PAGE_VIEW_UPDATE_FAILED	-6513	Could not update the PageView Control.
IGE_GUI_PAGE_VIEW_ENABLE_FAILED	-6512	Could not enable the PageView Control.
IGE_GUI_DISPLAY_ZOOM_CHANGE_FAILED	-6511	Could not modify the properties of the Display Zoom object for this image display.
IGE_GUI_DISPLAY_ZOOM_SET_OBJ_FAILED	-6510	Could not set the Display Zoom object for this image display.
IGE_GUI_DISPLAY_ZOOM_GET_OBJ_FAILED	-6509	Could not retrieve the Display Zoom object for this image display.
IGE_GUI_LAYOUT_CHANGE_FAILED	-6508	Could not modify the properties of the Display Layout object for this image display.
IGE_GUI_LAYOUT_GET_OBJ_FAILED	-6507	Could not retrieve the DIsplay Layout object for this image display.
IGE_GUI_PAGE_DISPLAY_SET_OBJ_FAILED	-6506	Could not set the PageDisplay object for this image's PageView Control.
IGE_GUI_PAGE_DISPLAY_CREATE_OBJ_FAILED	-6505	The CreatePageDIsplay() call failed.
IGE_GUI_NO_PAGE	-6504	Invalid IGPage object.
IGE_GUI_NO_PAGE_DISPLAY	-6503	Invalid IGPageDisplay object.
IGE_GUI_NO_PROCESSING_CTL	-6502	The control is not associated with a valid IGProcessingCtl control.
IGE_GUI_NO_DISPLAY_CTL	-6501	The control is not associated with a valid IGDisplayCtl control.

IGE_GUI_NO_FORMATS_CTL	-6500	The control is not associated with a valid IGFormatsCtl control.
IGE_UNREGISTER_CLASS_FAILED	-6040	Microsoft Windows function: UnregisterClass() failed.
IGE_INVALID_WINDOW	-6030	An invalid window handle was passed as one of the parameters to the function.
IGE_WINDOW_NOT_ASSOCATED	-6020	An attempt was made to disassociate a window that was never associated.
IGE_CREATE_WINDOW_FAILED	-6010	Microsoft Windows function: `CreateWindow()` failed.
IGE_REGISTER_CLASS_FAILED	-6000	Microsoft Windows function: `RegisterClass()` failed.

PDF/PostScript Component Error Codes

PDF Objects

IGE_PDF_STREAM_INVALID	-5807	Invalid stream object
IGE_PDE_OBJECT_INVALID	-5806	Invalid PDE object
IGE_PDF_COLORSPACE_INVALID	-5805	Invalid color space object
IGE_PDF_ELEMENT_INVALID	-5804	Invalid element
IGE_PDF_CONTENT_CANT_SET	-5803	Cannot set content
IGE_PDF_CONTENT_INVALID	-5802	Invalid content object
IGE_PDF_OBJECT_INVALID	-5801	Invalid PDF object

PDF Page

IGE_PDF_PAGE_CANT_DELETE	-5706	Cannot delete page.
IGE_PDF_PAGE_CANT_INSERT	-5705	Cannot insert page.
IGE_PDF_PAGE_CANT_SAVE	-5704	Cannot save page.
IGE_PDF_PAGE_CANT_OPEN	-5703	Cannot open page.
IGE_PDF_PAGE_CANT_CREATE	-5702	Cannot create page.
IGE_PDF_PAGE_INVALID	-5701	Invalid page object

PDF Document

IGE_PDF_DOC_NOT_AUTHORIZED	-5605	This operation is not permitted
IGE_PDF_DOC_CANT_SAVE	-5604	Cannot save document
IGE_PDF_DOC_CANT_OPEN	-5603	Cannot open document
IGE_PDF_DOC_CANT_CREATE	-5602	Cannot create document

IGE_PDF_DOC_INVALID	-5601	Invalid document object

PDF General

IGE_PDF_PRINTING_FAILED	-5541	Printing failed
IGE_PDF_CANT_CREATE_OBJECT	-5540	Cannot create object
IGE_PDF_CANT_EXTRACT_TEXT	-5539	Cannot extract text
IGE_PDF_INVALID_BITCOUNT	-5538	Invalid bit count
IGE_PDF_INVALID_COMPRESSION	-5537	Invalid compression
IGE_PDF_INVALID_PARAMETER	-5536	Invalid parameter
IGE_PDF_CANT_CONVERT_DATA	-5535	Rasterization failed
IGE_PDF_CANT_WRITE_DATA	-5534	Cannot write PDF stream
IGE_PDF_CANT_READ_DATA	-5533	Cannot read PDF stream
IGE_PDF_NOT_INITIALIZED	-5532	PDF engine is not initialized
IGE_PDF_FAILURE	-5531	General PDF error

PostScript General

IGE_PS_POSTSCRIPT_ERROR	-5504	The file contains a PostScript error
IGE_PS_EMPTY_JOB	-5503	Empty PostScript job
IGE_PS_NOT_INITIALIZED	-5502	PostScript engine is not initialized
IGE_PS_FAILURE	-5501	General PostScript job error

Multipage Error Codes

IGE_MP_IMAGE_NOT_ASSOCIATED	-5404	Multipage image is not opened
IGE_MP_ASSOCIATE_FAILURE	-5403	Multipage image open failure
IGE_MP_CANT_OPEN_MUTEX	-5402	Synchronization error
IGE_MP_IMVALID_PAGE_NUMBER	-5401	Invalid page number
IGE_MP_INVALID_HANDLE	-5400	Invalid multipage image handle

PS2 TEXT Function Error Codes

IGE_PS2TEXT_ERROR	-5350	GhostScript PS2TEXT error

PDF Read Function Error Codes

IGE_PDFREAD_GENERAL	-5301	GhostScript general error

IGE_PDFREAD_PSERROR	-5300	GhostScript PostScript error

Display Error Codes

IGE_FAIL_TO_DRAW_IMAGE	-4910	cannot draw image
IGE_INVALID_PRM_OPERATION	-4890	invalid operation
IGE_INVALID_PARAMETER	-4880	invalid parameter
IGE_INVALID_IMAGE_HANDLE	-4870	invalid image handle
IGE_INVALID_DEVICE_CONTEXT	-4860	invalid device context
IGE_INVALID_DISPLAY_PARAMETER	-4850	invalid display parameter
IGE_FORMAT_OPERATION_IS_NOT_SUPPORTED	-4840	format operation is not supported
IGE_INVALID_FORMAT_BIT_COUNT	-4835	invalid format bit count
IGE_INVALID_FORMAT_HEADER	-4830	invalid format header
IGE_INVALID_FORMAT_METHOD_ID	-4825	invalid format method ID
IGE_FAIL_TO_ALLOC_MUTEX	-4820	cannot allocate critical session data
IGE_GROUP_IS_NOT_EXIST	-4815	group does not exist
IGE_INVALID_OBJECT	-4811	invalid object
IGE_THREAD_ALREADY_ASSOCIATED	-4810	thread is already associated
IGE_FAIL_TO_ALLOC_PAMETER_GROUP	-4805	cannot allocate parameter group
IGE_CANT_OPEN_FORMAT_STORAGE_MUTEX	-4801	cannot start critical session
IGE_CANT_OPEN_PARAMETER_MUTEX	-4800	cannot start critical session

Component related error codes

IGE_FUNCTION_DOESNT_EXIST	-3702	The function doesn't exist in the specified component
IGE_COMPONENT_NOT_ATTACHED	-3701	The component isn't attached
IGE_CANT_ATTACH_COMP	-3700	The specified component can't be attached

Auto Detect Error codes

IGE_FILE_IS_UNSUPPORTED_FILE	-3605	File format is not supported
IGE_FILE_IS_HLP	-3604	File is a Microsoft sytem Help file
IGE_FILE_IS_DOC	-3603	File is a Microsoft DOC file
IGE_FILE_IS_ZIP	-3602	File is a PKZIP file
IGE_FILE_IS_EXE	-3601	File is a EXE, DLL, DRV, FNT, OCX, or 386

IGE_FILE_IS_SYSTEM_FILE	-3600	The image file passed in is really one of the following system files and not an image

Batch conversion CB Error codes

IGE_BATCH_WRONG_CB_TYPE	-3500	Wrong CB type passed to IG_batch_CB_register

Disk File Access Error codes

IGE_FILE_INVALID_PATH	-3450	Invalid path
IGE_FILE_INVALID_FILENAME	-3443	Invalid file name
IGE_FILE_CANT_DELETE	-3442	Cannot delete file
IGE_FILE_CANT_SAVE	-3441	Cannot save file
IGE_FILE_CANT_OPEN	-3440	Cannot open file

Image Processing Error codes

IGE_INVALID_RANK_TYPE	-3424	Specified type of rank in the rank filter is invalid.
IGE_INVALID_RESOLUTION_VALUE	-3423	Resolution value is not supported by a file format.
IGE_INVALID_ALPHA_CHANNEL	-3422	It is possible that width or height not equal to original image.
IGE_INVALID_CLIPING_RECT	-3421	Invalid clipping rect
IGE_RASTER_LINE_INVALID	-3420	The given raster line is invalid, it should be between 0 and the height of the image -1
IGE_PROC_INVAL_FOR_8BIT_INDEXED	-3410	Proc does not work on the 8i images
IGE_DIB_DIMENSIONS_ARE_INVALID	-3400	Either height or width of the DIB is wrong.
IGE_INVALID_BIT_MASK	-3390	The red, green and blue components of the bit mask overlap.
IGE_POINTER_IS_NULL	-3380	Pointer passed to an IP function is NULL, but it would point at object
IGE_INVALID_RESOLUTION_UNIT	-3370	The units of the image resolution are not supported
IGE_PROC_REQUIRE_8BIT_GRAYSCALE	-3360	This function can be used on 8-bit grayscale images only

IGE_PASSWORD_LENGTH_INVALID	-3350	Password must be at least 1-byte long (should be at least 5 bytes)
IGE_INVALID_ENCRYPT_MODE	-3340	The selected Encryption Method is invalid
IGE_INVALID_POINTER	-3330	ApointerwasdetectedtobeNULL
IGE_PARAMETER_OUT_OF_LIMITS	-3320	A parameter is out of its legal range
IGE_PROC_INVAL_FOR_PALETTE_IMG	-3310	This function can not be used on 8-bit color images - try to promote to 24-bit
IGE_PROC_INVAL_FOR_BIT_COUNT	-3300	This function can not be used on imageswiththisone'sbitcount
IGE_DIB_POINTER_IS_NULL	-3220	The DIB pointer about to be used is NULL (invalid)
IGE_COLOR_SPACE_INVALID	-3210	Invalid type of color space
IGE_TILE_IS_LARGER_THAN_IMAGE	-3200	The tile image must be the same size or smaller in both dimensions than the original source
IGE_INVALID_SKEW_POINTS	-3190	A valid line could not be drawn through the two point provided. Y1==Y2
IGE_INVALID_SIGMA	-3180	Invalid sigma
IGE_INVALID_KERN_NORMALIZER	-3170	Invalid kernel normalizer
IGE_INVALID_NOISE_TYPE	-3160	Invalid noise type
IGE_INVALID_KERN_MOTION_EXTENT	-3150	Invalid kernel motion extent
IGE_INVALID_LUT_ARITH_FUNC	-3140	The LUT_ARITH_FUNC is not a valid function number, check the constant
IGE_INVALID_RAMP_DIRECTION	-3130	The contrast ramps direction is not supported
IGE_REQUIRES_CONVEX_ROI	-3120	This function requires a convex ROI. The one passed in must be convex
IGE_ROI_WRONG_TYPE	-3110	Region of interest is wrong type
IGE_DIB_HAS_NO_PALETTE	-3101	The image passed in does not have a palette and this routine requires one
IGE_DIB_BIT_COUNTS_NOT_EQUAL	-3100	The images must have the same bit count
IGE_DIB_DIMENSIONS_NOT_EQUAL	-3090	The images must be the same dimensions
IGE_INVALID_SIGMA	-3080	Invalid sigma
IGE_DIBS_ARE_INCOMPATIBLE	-3070	The images are not compatible for this function, either dimension, bit count, or both
IGE_AI_HANDLE_INVALID	-3060	This routine requires an AccuSoft handle. The handle passed in was not allocated by AccuSoft
IGE_AI_HANDLES_USED_UP	-3050	The maximum number of AccuSoft handles has been used - no more left. Free up some and try again

IGE_BAD_KERN_TYPE	-3040	Bad kernel type
IGE_FREE_FAILED	-3030	An internal memory free has failed, usually a bad handle, or corrupted system
IGE_ALLOC_FAILED	-3020	Memory required for this routine could not be allocated, free up some resources and try again
IGE_LOCK_FAILED	-3010	Memory required for this routine could not be locked, most likely running out of memory resources
IGE_WRONG_DIB_BIT_COUNT	-3000	The DIB has bit with the wrong bit count for this routine

Sync Error Codes

IGE_CANT_UNLOCK_DATA	-2904	Can't unlock data
IGE_CANT_LOCK_DATA	-2903	Can't lock data.
IGE_CANT_UNLOCK_IMAGE	-2902	Can't unlock image
IGE_CANT_LOCK_IMAGE	-2901	Can't lock image.
IGE_IMAGE_IS_LOCKED	-2900	Image is locked

TIFF Filter Specific Errors

IGE_END_OF_IMAGE	-2610	
IGE_CANNT_OPEN_INTERNET_SESSION	-2600	Can't open internet session
IGE_CANNT_OPEN_INTERNET_CONNECTION	-2590	Can't open internet connection
IGE_CANNT_OPEN_TEMPORARY_FILE	-2580	Can't open temporary file
IGE_CANNT_OPEN_GOPHER_FILE	-2570	Can't open Gopher file
IGE_CANNT_OPEN_HTTP_FILE	-2560	Can't open HTTP file
IGE_CANNT_OPEN_FTP_FILE	-2550	Can't open FTP file
IGE_JBIG_STREAM_OPEN_FAILURE	-2540	JBIG support library returned an error in return code
IGE_TIF_INVALID_CLASS_F_IMAGE	-2530	Error writing TIF class F format
IGE_ICA_IBM_MMR_COMP_ERROR	-2520	Error in IBM MMR IOCAMO:DCA compression
IGE_ICA_COMP_NOT_SUPPORTED	-2510	MO:DCAIOCA Compression is not supported at this time
IGE_CLP_INVALID_FORMAT_ID	-2500	Windows clipboard file contains an unsupport Format ID at this page
IGE_JBG_IMG_CNTRL_NOT_FOUND	-2486	JBIG Extension Image Control not found for save
IGE_JBIG_EXTENSION_NOT_LOADED	-2485	JBIG Extension has not been loaded
IGE_ABIC_EXTENSION_NOT_LOADED	-2484	ABIC Extension has not been loaded

IGE_IMBEDDED_IMAGE_FAILURE	-2483	Failure occurred while reading a file format imbedded in another
IGE_RASTER_WRITE_FAILURE	-2482	Unable to write Raster to Output Device (Full Device)
IGE_TILE_NOT_PRESENT	-2481	Tile is not present
IGE_LZW_EXTENSION_NOT_LOADED	-2480	LZW Extension has not been loaded
IGE_PRIV_TAG_TYPE_INVALID	-2479	private user tag had an invalid type
IGE_REPAGE_FAILED	-2478	unable to write new page numbers while repaging file
IGE_INVALID_COMP_BIT_DEPTH	-2477	bit depth is invalid for this compression scheme
IGE_INVALID_STRIP_BYTE_CNT	-2476	strip byte count was zero and could not be estimated
IGE_CORRUPTED_FILE	-2475	data in file was not what was expected and could not be interpreted
IGE_PAGE_COUNT_FAILURE	-2474	count not count the number of pages in the file
IGE_BAD_DATA_TYPE	-2473	Data type detected is not valid
IGE_INVALID_IMG_DEM	-2472	Image Dimension was invalid
IGE_LZW_ERROR	-2471	Error occured in LZW decode
IGE_RASTER_TO_LONG	-2470	One raster lines exceeds the max num of bytes
IGE_PLANAR_CONFIG_ERROR	-2469	Planar Config detected is unsupported
IGE_IMAGE_DATA_WRITE_ERROR	-2468	Was not able to write IMAGE data to the IOS
IGE_TAG_WRITE_ERROR	-2467	Was not able to write TAG info to the IOS
IGE_IFD_WRITE_ERROR	-2466	Was not able to write IFD info to the IOS
IGE_CANT_REALLOC_MEM	-2465	Was not able to reallocate memory
IGE_DIB_GET_FAILURE	-2464	Was unable to retreive the DIB information
IGE_HEADER_WRITE_FAILED	-2463	Header write failed
IGE_IMAGE_DATA_READ_ERROR	-2462	Was unable to read all image data requested
IGE_RASTER_FEED_ERROR	-2461	Error feeding raster data to the DIB
IGE_COMP_NOT_SUPPORTED	-2460	Compression is not supported at this time
IGE_INVALID_REQ_INFO	-2459	Required information was not supplied
IGE_INVALID_PHOTOMETRIC	-2458	Photometric tag value was found to be invalid
IGE_INVALID_COLOR_MAP	-2457	Color Map was found to be invalid
IGE_INVALID_BITS_PER_SAMPLE	-2456	Bits per sample tag was invalid

IGE_CANT_READ_TAG_DATA	-2455	Was unable to read all TAG information
IGE_INVALID_BYTE_ORDER	-2454	Byte order flag was not Intel or Motorola
IGE_SEEK_FAILURE	-2453	IOS position seek failed
IGE_IFD_PROC_FAILURE	-2452	Invalid IFD information was detected
IGE_INVALID_IFD	-2451	IFD Read did not contain correct number of bytes
IGE_INVALID_TAG	-2450	Tag Read did not contain correct number of bytes

Miscellaneous Error Codes

IGE_X_NULL_DISPLAY	-2448	X display specified is NULL. ASCII filter is unable to draw text.
IGE_FILTER_UNKNOWN_FORMAT	-2447	Unknown file format
IGE_FILTER_CANT_GET_INFOFUNC	-2446	Fail to get filter's info function
IGE_DIB_RES_UNITS_NOT_SUPPORTED	-2445	DIB resolution units is not supported
IGE_UNDEFNIED_COLOR_SPACE_ID	-2444	color space ID is not defined
IGE_NOT_SUPPORTED_COMP	-2443	Compression is not supported at this time
IGE_PNG_NO_IDAT_CHUNK	-2442	failed to READ a manditory IDAT Chunk
IGE_PNG_READ_FAILED	-2441	failed to READ png data
IGE_PNG_CHUNK_READ_FAILED	-2440	failed to READ the correct number of bytes for png chunk
IGE_PNG_WRITE_FAILED	-2439	failed to write png data
IGE_PNG_CHUNK_WRITE_FAILED	-2438	failed to write the correct number of bytes for png chunk
IGE_STANDARD_FEATURE	-2437	This feature is not available in Lite version
IGE_LOAD_FUNCTION_GET_FAILED	-2436	Was not able to intialize the filer load function
IGE_ALLOC_SELECTOR_FAILED	-2435	AllocSelector() failed, couldn't get an entry into the Global Descriptor able
IGE_CAN_NOT_OPEN_TEMP_FILE	-2434	The temporary file need for this function could not be openedcreated
IGE_PROC_INVAL_FOR_RUNS_DIB	-2433	This function can not be used on DIBs in the Runs format - convert first IG_IP_convert_runs_to_DIB
IGE_OPERATION_IS_NOT_ALLOWED	-2432	
IGE_PJPEG_NOT_SUPPORTED	-2427	Progressive JPEG feature is not supported
IGE_JPG_INVALID_HUFFMAN_TABLE	-2426	Invalid huffman table

IGE_JPG_INVALID_HUFFMAN_ID	-2425	Invalid huffman table descriptor
IGE_JPG_INVALID_QTABLE_PRECISION	-2424	Invalid quantization table precision
IGE_JPG_INVALID_QTABLE_ID	-2423	Invalid quantization table descriptor
IGE_JPG_UNRECOGNIZE	-2422	Unrecognized JPEG marker encountered
IGE_PJPEG_INVALID_SCAN_COUNT	-2421	Invalid number of scans for progresive JPEG write
IGE_PJPEG_INVALID_SCAN_CONFIGURATION	-2420	Invalid configuration of scans for progressive JPEG write
IGE_IMNET_INVALID_WIDTH	-2418	Invalid width for IMNET
IGE_NAME_CONV_NOT_SUPPORTED	-2416	Batch Naming configuration not supported
IGE_LOG_FILE_CREATE_ERROR	-2415	Unable to create Batch Log File
IGE_DIRECTORY_CREATE_ERROR	-2414	Unable to create Destination Directory
IGE_BITDEPTH_NOTSUPPORTED	-2413	This Bit-Depth is not supported for this write format
IGE_G3_PREMATURE_EOL_AT_SCAN_LINE	-2412	Group 3 premature EOL
IGE_G3_BAD_1D_CODE_AT_SCAN_LINE	-2411	Group 3 bad 1D code
IGE_G3_PREMATURE_EOF_AT_SCAN_LINE	-2410	Group 3 premature EOF
IGE_G4_BAD_DECODING_STATE_AT_SCAN_LINE	-2403	Group 4 bad decoding state
IGE_G4_BAD_2D_CODE_AT_SCAN_LINE	-2402	Group 4 invalid 2D code
IGE_G4_PREMATURE_EOL_AT_SCAN_LINE	-2401	Group 4 premature EOL
IGE_G4_PREMATURE_EOF_AT_SCAN_LINE	-2400	Group 4 premature EOF
IGE_PALETTE_FILE_INVALID_HALO_PAL	-2399	Detected Dr. Halo Palette file is invalid
IGE_PALETTE_FILE_NOT_DETECTED	-2398	The file is not a valid palette file
IGE_PALETTE_FILE_READ_ERROR	-2397	Error reading from a palette file
IGE_PALETTE_FILE_WRITE_ERROR	-2396	Error writing to a palette file
IGE_PALETTE_FILE_TYPE_INVALID	-2395	IG_PALETTE_ value is not known
IGE_NO_BITMAP_FOUND	-2394	WPG, WMF etc. No raster image exists in file
IGE_CANT_WRITE_FILE	-2393	File can't be saved in the specified format.
IGE_EPS_NO_PREVIEW	-2392	EPS file has no screen preview image to load
IGE_BAD_FILE_FORMAT	-2391	Format is not correct.
IGE_NO_BITMAP_REGION	-2390	No bitmap region
IGE_INTERRUPTED_BY_USER	-2113	Status bar callback returned FALSE
IGE_INVALID_BITCOUNT_FOR_FORMAT	-2112	The bitcount found is not supported by this format
IGE_FILE_FORMAT_IS_READONLY	-2111	The image file is read only and can not be written to
IGE_INVALID_IMAGE_FORMAT	-2110	The image file is invalid as the expected format

IGE_CANT_READ_FILE	-2100	Can't read file
IGE_CANT_SETUP_DIB_HEADER	-2095	Can't setup DIB header
IGE_INSTANCE_CLEANUP_ERROR	-2091	Instance cleanup error
IGE_INSTANCE_FAILURE	-2090	Instance failure
IGE_INVALID_COMPRESSION	-2080	Invalid compression
IGE_FORMAT_NOT_DETECTABLE	-2073	Save format can not be detected from file extension used
IGE_CANT_WRITE_HEADER	-2072	Can't write header
IGE_CANT_WRITE_PIXELS	-2071	Can't write pixel data
IGE_CANT_WRITE_PALETTE	-2070	Can't write palette
IGE_INVALID_HEADER	-2061	Invalild file header
IGE_INVALID_FILE_TYPE	-2060	Invalid file type
IGE_CANT_READ_HEADER	-2052	Can't read header
IGE_CANT_READ_PIXELS	-2051	Can't read pixel data
IGE_CANT_READ_PALETTE	-2050	Can't read palette
IGE_FILE_SIZE_WRITE_ERROR	-2041	Could not write file size field to BMP
IGE_BMP_IS_COMPRESSED	-2040	The BMP file is in compressed (RLE) format
IGE_FILE_TO_SMALL_TO_BE_BMIH	-2035	The file is too small to be a valid BMIH
IGE_FILE_IS_NOT_BMP	-2034	The BMFH Magic number is invalid
IGE_FILE_TO_SMALL_TO_BE_BMFH	-2033	The file is too small to be a valid BMFH
IGE_FILE_CANT_BE_CLOSED	-2032	An attempt to close a file failed
IGE_FILE_CANT_BE_CREATED	-2031	An attempt to create a file failed, it may already exist in the provided path
IGE_FILE_CANT_BE_OPENED	-2030	An attempt to open a file failed, it may not exist in the provided path
IGE_CANT_DETECT_FORMAT	-2010	The format of the file can not be determined
IGE_PAGE_COULD_NOT_BE_READ	-2007	The page number could not be determined
IGE_PAGE_INVALID	-2006	The page number provided is outside of the range of valid pages for this file
IGE_PAGE_NOT_PRESENT	-2005	The requested image page does not exist in the file
IGE_THUMBNAIL_NOT_SUPPORTED	-2002	Thumbnails are not supported by this format
IGE_THUMBNAIL_READ_ERROR	-2001	A read error occured while reading a thumbnail
IGE_THUMBNAIL_NOT_PRESENT	-2000	Thumnails are supported but none can be found in this image file
IGE_PASSWORD_INVALID	-1016	The Password is not recognized

IGE_INVALID_PATTERN_BITMAP	-1015	The DDB was > 1 bit per pixel or the width was > 8 or the height was > 8
IGE_COULD_DISPLAY_DDB	-1014	BitBlt() failed. Couldn't display the DDB
IGE_CREATE_BITMAP_FAILED	-1013	CreateBitmap() failed. Couldn't create a DDB.
IGE_COULD_NOT_GET_DDB_BITS	-1012	GetDIBits() failed. Couldn't get the DDB's image data
IGE_COULD_NOT_GET_DDB_DIMENSIONS	-1011	GetObject() failed. Couldn't get the DDB's dimensions
IGE_SETCLIPBOARDDATA_FAILED	-1010	Could not put data into the clipboard
IGE_CLIPBOARD_OPEN_FAILED	-1009	Could not open the clipboard
IGE_NO_CLIPBOARD_IMAGE_AVAILABLE	-1008	No image is available for a clipboard paste
IGE_INVALID_RECTANGLE	-1007	Occurs when a rectangle's left >= right or top >= bottom
IGE_INTERNAL_ERROR	-1005	An internal error has occured, contact ACCUSOFT techical support
IGE_INVALID_STANDARD_KERNEL	-1004	The kernel expected one of the predefined ones, yours could not be found
IGE_EVAL_DLL_TIMEOUT_HAS_EXPIRED	-1003	The DLL is an Evaluation copy and as timed out - contact ACCUSOFT to purchase a release copy
IGE_OUT_OF_MEMORY	-1000	No more global memory is available for allocation, reduced used resources
IGE_FUNCTIONALITY_NOT_SUPPORTED	-900	The ImageGear functionality is not supported under this platform.
IGE_EXTENSION_INITIALIZATION_FAILED	-800	The specified ImageGear extension was unable to initialize.
IGE_INVALID_EXTENSION_MODULE	-700	The specified ImageGear extension file was not a valid extension file
IGE_INVALID_CONTROL_OPTION	-600	Invalid image control option ID
IGE_COMPONENT_ATTACH_FAILURE	-553	Failure to attach component.
IGE_INVALID_COMPONENT_MODULE	-552	The module is not component or has wrong interface
IGE_INVALID_FUNCTION_POINTER	-551	Invalid pointer to the component's function
IGE_INVALID_FUNCTION_NAME	-550	The name of the component's function is invalid
IGE_FILTER_CTRL_INVALID_NAME	-512	The name of control parameter is invalid or not supported
IGE_INVALID_FILTER_OPERATION	-511	Requested operation is not supported by format filter
IGE_FILTER_NOT_LOADED	-510	Requested format filter is not present

IGE_EXTENSION_NOT_LOADED	-500	The ImageGear extension was not present or couldn't be loaded
IGE_BUFFER_HAS_INSUFFICIENT_SIZE	-404	Incoming buffer has insufficient size
IGE_PARAMETER_HAS_INVALID_VALUE	-401	Incoming parameter is invalid
IGE_ERROR_COMPRESSION	-400	
IGE_NOT_SUPPORTED_BY_PLATFORM	-350	The last function used is not supported by this platform.
IGE_NOT_LITE_FEATURE	-301	An attempt was made to use a Non IG-Lite feature in the IG-Lite product.
IGE_NOT_IMPLEMENTED	-200	For internal reference of areas to return to
IGE_NOT_DONE_YET	-100	For internal reference of areas to return to
IGE_FAILURE	-1	General error - Failure
IGE_SUCCESS	0	No errors -- Success

Appendix B: Non-Image Data Storage

This Appendix provides you with detailed information about ImageGear:

- Metadata Structure “ValueType” and “Value”.
- TIFF Non-Image Data Structure
- JPEG Non-Image Data Structure
- EXIF-TIFF Non-Image Data Structure
- EXIF-JPEG Non-image Data Structure
- CIFF Non-Image Data Structure
- IPTC Non-Image Data Structure
- PNG Non-Image Data Structure
- GIF Non-image Data Structure
- DCRAW Non-Image Data Structure
- HD Photo Non-Image Data Structure

Metadata Structure “ValueType” and “Value”

Here are possible combinations of `ValueType` and `Value` elements of `AT_DATALIST_ITEM` ImageGear metadata structure:

ValueType	Value
`AM_TID_META_INT8`	■ The “FLTR.METADATA_FORMAT” global parameter value is “text” : Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 8-bit signed integers separated by comma. **Example**: “-128;0;+45;56” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” : Value is pointer to array of 8-bit signed integers of size “Length”

AM_TID_META_UINT8	■ The “FLTR.METADATA_FORMAT” global parameter value is “text” : Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 8-bit unsigned integers separated by comma. The hexadecimal values are allowed. **Example**: “0;+18;255;0xFA” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 8-bit signed integers of size “Length”
AM_TID_META_INT16	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 16-bit signed integers separated by comma. **Example**: “0;+1800;-255;32355” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 16-bit signed integers of size “Length”
AM_TID_META_UINT16	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 16-bit unsigned integers separated by comma. The hexadecimal values are allowed. **Example**: “0;+543;2550;0x12FF” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 16-bit signed integers of size “Length”
AM_TID_META_INT32	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 32-bit signed integers separated by comma. **Example**: “0;+67;-235987;32355” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit signed integers of size “Length”

AM_TID_META_UINT32	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 32-bit unsigned integers separated by comma. The hexadecimal values are allowed. **Example**: “0;+543;12362550;0x56FDE345” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit signed integers of size “Length”
AM_TID_META_INT64	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated string – textual representation of “Length” 64-bit signed integers separated by comma. ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit signed integers of size “Length”
AM_TID_META_UINT64	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 64-bit unsigned integers separated by comma. The hexadecimal values are allowed. ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit signed integers of size “Length”
AM_TID_META_BOOL	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NULL-terminated string – textual representation of “Length” Boolean values separated by comma. **Example**: “TRUE;FALSE;false;true” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit unsigned integers of size “Length” The value 1 represents boolean TRUE and the value 0 represents boolean FALSE.

AM_TID_META_RATIONAL_INT32	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” rational values separated by comma. Each rational value is fraction where both numerator and denominator are 32-bit signed integers separated by slash. Zero is allowed. **Example**: “+324/-567;0/0;-68/45668” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit signed integers of size 2 * “Length” Each two elements represent pair (numerator, denominator) that represents rational number numerator/denominator.
AM_TID_META_RATIONAL_UNT32	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” rational values separated by comma. Each rational value is fraction where both numerator and denominator are 32-bit unsigned integers separated by slash. Zero is allowed. **Example**: “+324/567;0/0” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 32-bit unsigned integers of size 2 * “Length” Each two elements represent pair (numerator, denominator) that represents rational number numerator/denominator.
AM_TID_META_FLOAT	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 4-byte float-point values separated by comma. **Example**: “-298.98676;3568732;6.9876E-10;0” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 4-byte float-point values of size “Length”

AM_TID_META_DOUBLE	■ The “FLTR.METADATA_FORMAT” global parameter value is “text”: Value is pointer to NUL-terminated ASCII string – textual representation of “Length” 8-byte (double precision) float-point values separated by comma. **Example**: “-29842.9867698098;356873245345;6.98766575489E+50;0” ■ The “FLTR.METADATA_FORMAT” global parameter value is “binary” (DLL only): Value is pointer to array of 8-byte (double precision) float-point values of size “Length”
AM_TID_META_STRING	■ It does not depend on “FLTR.METADATA_FORMAT” global parameter value Value is pointer to NUL-terminated ASCII string, “Length” – length of the string (NULL is not counted). **Example**: “Simple string”
AM_TID_RAW_DATA	■ It does not depend on “FLTR.METADATA_FORMAT” global parameter value Value is pointer to byte array that represent binary data, “Length” – size of the array in bytes.

Non-Image Data Structure

In this section the metadata items (and sequence of metadata items) is written in table form where metadata type constants are in the shorted form. For example the LEVEL_START should be treated as IG_METAD_LEVEL_START, UINT32 – as AM_TID_META_UINT32, RAW_DAT – as AM_TID_RAW_DATA etc.

See Also:

- TIFF Non-Image Data Structure
- JPEG Non-Image Data Structure
- EXIF-TIFF Non-Image Data Structure
- EXIF-JPEG Non-image Data Structure
- IPTC Non-Image Data Structure
- PNG Non-Image Data Structure

TIFF Non-Image Data Structure

TIFF metadata structure is framed in several levels:

- TIFF level containing all TIFF metadata
- IFD level,
- Tags

TIFF level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“TIFF”	IG_FORMAT_TIF	LEVEL_START	0	NULL	0	TRUE
“TIF_ HEADER”	59936	VALUE_ITEM	UINT16	0x4949 or 0x4D4D	1	TRUE
Main IFD level						
“TIFF”	IG_FORMAT_TIF	LEVEL_END	0	NULL	0	TRUE

All items between items with **Name** “TIFF” and **Id** IG_FORMAT_TIF (Type LEVEL_START and LEVEL_END) are interpreted as TIFF data. If during sending data from application level to filter level the first item is omitted the data will not be parsed and saved.

NOTE: The subIFDs is not parsed and passed.

IFD level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IFD”	0	LEVEL_START	0	NULL	0	TRUE
One or more tag items						
“IFD”	0	LEVEL_END	0	NULL	0	TRUE

Tags

Most of TIFF tags are passed through metadata callback as

Name	Id	Type	Value Type	Value	Value Length	Read Only

<TagName>	<TagId>	VALUE _ITEM	<Value Type>	<TagValue>	<TagCount>	<R.o.>

The valid values of <TagName>, <TagId> and “Read Only” attributes are available in Description of TIFF tags Table.

See also: TIFF Complex Data Tags

NOTE: It is possible to use tag identifier and value of nonstandard tag (user defined tag).

The <TagCount> is real tag count value that is read/written in file.

The < ValueType > value is the metadata type constant that matches TIFF tag type as described in the table below:

TIFF tag type	Metadata type
BYTE	AM_TID_META_UINT8
SBYTE	AM_TID_META_INT8
SHORT	AM_TID_META_UINT16
SSHORT	AM_TID_META_INT16
LONG	AM_TID_META_UINT32
SLONG	AM_TID_META_INT32
RATIONAL	AM_TID_META_RATIONAL_UINT32
SRATIONAL	AM_TID_META_RATIONAL_INT32
FLOAT	AM_TID_META_FLOAT
DOUBLE	AM_TID_META_DOUBLE
ASCII	AM_TID_META_STRING
UNDEFINED	AM_TID_RAW_DATA

Description of TIFF tags

Item Name	Item Id	Read only
“NewSubfileType”	254	TRUE
“SubfileType”	255	TRUE
“ImageWidth”	256	TRUE

“ImageLength”	257	TRUE
“BitsPerSample”	258	TRUE
“Compression”	259	TRUE
“PhotometricInterpretation”	262	TRUE
“Threshholding”	263	FALSE
“CellWidth”	264	TRUE
“CellLength”	265	TRUE
“FillOrder”	266	FALSE
“DocumentName”	269	FALSE
“ImageDescription”	270	FALSE
“Make”	271	FALSE
“Model”	272	FALSE
“StripOffsets”	273	TRUE
“Orientation”	274	FALSE
“SamplesPerPixel”	277	TRUE
“RowsPerStrip”	278	TRUE
“StripByteCounts”	279	TRUE
“MinSampleValue”	280	FALSE
“MaxSampleValue”	281	FALSE
“XResolution”	282	FALSE
“YResolution”	283	FALSE
“PlanarConfiguration”	284	TRUE
“PageName”	285	FALSE
“XPosition”	286	FALSE
“YPosition”	287	FALSE
“FreeOffsets”	288	TRUE
“FreeByteCounts”	289	TRUE
“GrayResponseUnit”	290	FALSE
“GrayResponseCurve”	291	FALSE
“T4Options”	292	TRUE

“T6Options”	293	TRUE
“ResolutionUnit”	296	FALSE
“PageNumber”	297	TRUE
“TransferFunction”	301	FALSE
“Software”	305	FALSE
“DateTime”	306	FALSE
“Artist”	315	FALSE
“HostComputer”	316	FALSE
“Predictor”	317	TRUE
“WhitePoint”	318	FALSE
“PrimaryChromaticities”	319	FALSE
“ColorMap”	320	TRUE
“HalftoneHints”	321	FALSE
“TileWidth”	322	TRUE
“TileLength”	323	TRUE
“TileOffsets”	324	TRUE
“TileByteCounts”	325	TRUE
“InkSet”	332	FALSE
“InkNames”	333	FALSE
“NumberOfInks”	334	FALSE
“DotRange”	336	FALSE
“TargetPrinter”	337	FALSE
“ExtraSamples”	338	TRUE
“SampleFormat”	339	FALSE
“SMinSampleValue”	340	FALSE
“SMaxSampleValue”	341	FALSE
“TransferRange”	342	FALSE
“JPEGTables”	347	TRUE
“JPEGProc”	512	TRUE
“JPEGInterchangeFormat”	513	TRUE

“JPEGInterchangeFormatLngth”	514	TRUE
“JPEGRestartInterval”	515	TRUE
“JPEGLosslessPredictors”	517	TRUE
“JPEGPointTransforms”	518	TRUE
“JPEGQTables”	519	TRUE
“JPEGDCTables”	520	TRUE
“JPEGACTables”	521	TRUE
“YCbCrCoefficients”	529	FALSE
“YCbCrSubSampling”	530	TRUE
“YCbCrPositioning”	531	FALSE
“ReferenceBlackWhite”	532	FALSE
“Copyright”	33432	FALSE
“PhotoshopResources”	34377	FALSE
“ExifIFDPointer”	34665	TRUE
“GPSInfoIFDPointer”	34675	TRUE

NOTE: Please see file `metadata.h` for the full list of TIFF Metadata **Item Name** and **Item Id** constants.

TIFF Complex Data Tags

Some tags, which value is a complex data, can be passed in parsed form as following:

Name	Id	Type	Value Type	Value	Value Length	Read Only
<TagName>	<TagId>	LEVEL_START	0	NULL	0	TRUE
“TagType”	<TagId>	VALUE_ITEM	UINT16	<TagType>	1	FALSE
“TagCount”	<TagId>	VALUE_ITEM	UINT32	<TagCount>	1	FALSE
<Tag Value Block>: one or more metadata items that identify the tag value						
<TagName>	<TagId>	LEVEL_END	0	NULL	0	TRUE

The **<TagName>** and **<TagId>** values are the tag name and identifier (available values see in the Table Photoshop Image Resource metadata structure). It is possible to use the tag identifier and value of nonstandard tag (user defined tag).

The **<TagType>** value is the standard TIFF tag type constant described in TIFF 6.0 specification.

Currently only TIFF tag with ID 34377, where Adobe Photoshop and some other TIFF writers save image recourses (IPTC data, resolution, some LUT etc.), is passed in this form. The **<Tag Value Block>** of these TIFF tag has structure described in the Table Photoshop Image Resource metadata structure below.

Photoshop Image Resource metadata structure

Name	Id	Type	Value Type	Value	Value Length	Read Only
“PhotoshopImageResource”	<Res. Id>	LEVEL_START	0	NULL	0	TRUE
“PhotoshopImageResourceSize”	<Res. Id>	VALUE_ITEM	UINT16		1	TRUE
Resource data						
“DATA”	<Res. Id>	VALUE_ITEM	RAW_DATA		Variable	FALSE
Or in case the data are parsed (currently it is happen only if the resource is IPTC data)						
“IPTC”	0x1C00	LEVEL_START	0	NULL	0	TRUE
…						
“IPTC”	0x1C00	LEVEL_END	0	NULL	0	TRUE
“PhotoshopImageResource”	<Res. Id>	LEVEL_END	0	NULL	0	TRUE

Where the <Res. Id> is the Adobe Photoshop image resource identifier (see Adobe Photoshop SDK).

IPTC PhotoshopImageResource identifier is 0x0404.

Callback required for writing TIFF metadata items

Value of TIFF metadata tags that are in the table below can be changed using **LPAFT_IG_METAD_ITEM_SET_CB** callback only:

Item Name	Item Id
“XResolution”	282
“YResolution”	283
“ResolutionUnit”	296
“PageNumber”	297
“Software”	305
“DateTime”	306
“Artist”	315
“FillOrder”	266
“DocumentName”	269

All the rest TIFF metadata tags values can be written using **LPAFT_IG_METAD_ITEM_ADD_CB** callback only.

TIFF Metadata Item Name and ID constants

NOTE: Please see file `metadata.h` for the full list of TIFF Metadata **Item Name** and **Item Id** constants.

JPEG Non-Image Data Structure

JPEG metadata structure is framed in several levels:

- JPEG level containing all JPEG metadata,
- JPEG marker segment levels containing metadata of each separated JPEG marker segment.

JPEG level

Name	Id	Type	Value Type	Value	Value Length	Read Only

“JPEG-JFIF”	IG_FORMAT_JPG	LEVEL_START	0	NULL	0	TRUE
JPEG marker segment levels						
“JPEG-JFIF”	IG_FORMAT_JPG	LEVEL_END	0	NULL	0	TRUE

All items between items with Name “JPEG-JFIF” and Id IG_FORMAT_JPG (Type LEVEL_START and LEVEL_END) are interpreted as JPEG metadata. If during sending data from application level to filter level the first item is omitted the data will not be parsed and saved.

JPEG marker segment levels

The following JPEG marker segments metadata are supported:

- Frame marker segment level – SOF0, SOF1, SOF2 and SOF3 (read only),
- Scan marker segment level – SOS (read only),
- Define-quantization-table marker segment level – DQT (read only),
- Define-Huffman-tables marker segment level – DHT (read only),
- Comment marker segment level – COM (read/write),
- Application marker segment level – APP0-APP15 (read/write).

Frame marker segment level

Name	Id	Type	Value Type	Value	Value Length	Read Only
<Frame marker name>	<Frame marker Id>	LEVEL_START	0	NULL	0	TRUE
<Frame marker size tag name>	<Frame marker size tag Id>	VALUE_ITEM	UINT16		1	TRUE
“Precision”	212	VALUE_ITEM	UINT8		1	TRUE
“Lines”	213	VALUE_ITEM	UINT16		1	TRUE
“SamplesPerLine”	214	VALUE_ITEM	UINT16		1	TRUE
“ComponentNumber”	215	VALUE_ITEM	UINT8	<n>	1	TRUE
Frame component level 1						

Frame component level <n>						
<Frame marker name>	<Frame marker Id>	LEVEL_END	0	NULL	0	TRUE

<Frame marker Id>, **<Frame marker name>**, **<Frame marker size tag name>** and **<Frame marker size tag Id>** are described in the table below:

Frame Marker	<Frame marker name>	<Frame marker Id>	<Frame marker size tag name>	<Frame marker size tag Id>
SOF0	“SOF0”	0xFFC0	“SOF0_DATASIZE”	60192
SOF1	“SOF1”	0xFFC1	“SOF1_DATASIZE”	60193
SOF2	“SOF2”	0xFFC2	“SOF2_DATASIZE”	60194
SOF3	“SOF3”	0xFFC3	“SOF3_DATASIZE”	60195

Frame component level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“Component”	<Component No.>	LEVEL_START	0	NULL	0	TRUE
“Id”	<Component No.>	VALUE_ITEM	UINT		1	TRUE
“MCU_HV”	<Component No.>	VALUE_ITEM	UINT		1	TRUE
“QuantizationNumber”	<Component No.>	VALUE_ITEM	UINT		1	TRUE
“Component”	<Component No.>	LEVEL_END	0	NULL	0	TRUE

NOTE: **<Component No.>** is frame component identifier of appropriate component.

Scan marker segment level

Name	Id	Type	Value Type	Value	Value Length	Read Only

“SOS”	0xFFDA	LEVEL_START	0	NULL	0	TRUE
“SOS_DATASIZE”	60218	VALUE_ITEM	UINT16		1	TRUE
“ComponentNumber”	228	VALUE_ITEM	UINT8	<n>	1	TRUE
Scan component level 1						
Scan component level <n>						
“SpectralStart”	235	VALUE_ITEM	UINT8		1	TRUE
“SpectralEnd”	236	VALUE_ITEM	UINT8			TRUE
“AH_AL”	237	VALUE_ITEM	UINT8			TRUE
“SOS”	0xFFDA	LEVEL_START	0	NULL	0	TRUE

Scan component level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“Component”	<Component No.>	LEVEL_START	0	NULL	0	TRUE
“Selector”	<Component No.>	VALUE_ITEM	UINT8		1	TRUE
“DC_AC”	<Component No.>	VALUE_ITEM	UINT8		1	TRUE
“Component”	<Component No.>	LEVEL_END	0	NULL	0	TRUE

NOTE: **<Component No.>** is scan component selector of appropriate component.

Define-quantization-table marker segment level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“DQT”	0xFFDB	LEVEL_START	0	NULL	0	TRUE

“DQT_DATASIZE”	60219	VALUE_ITEM	UINT16		1	TRUE
“QuantizationTable”	<table 1 Id>	VALUE_ITEM	UINT8 or UINT16		64	TRUE
“QuantizationTable”	<table 1 Id>	VALUE_ITEM	UINT8 or UINT16		64	TRUE
“DQT”	0xFFDB	LEVEL_END	0	NULL	0	TRUE

NOTE: **<table Id >** is the appropriate quantization table identifier.

Define-Huffman-tables marker segment level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“DHT”	0xFFC4	LEVEL_START	0	NULL	0	TRUE
“DHT_DATASIZE”	60196	VALUE_ITEM	UINT16		1	TRUE
“DATA”	0xFFC4	VALUE_ITEM	RAW_DATA		<ValueSize>	TRUE
“DHT”	0xFFC4	LEVEL_END	0	NULL	0	TRUE

Comment marker segment level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“COM”	0xFFFE	LEVEL_START	0	NULL	0	TRUE
“COM_DATASIZE”	60254	VALUE_ITEM	UINT16		1	TRUE
“Comment”	0xFFFE	VALUE_ITEM	STRING		<ValueSize>	TRUE
“COM”	0xFFFE	LEVEL_END	0	NULL	0	TRUE

Application marker segment level

There are several application marker segments, which data structure is well known. These segments are parsed and their data are passed in special format.

These marker segments are:

- JFIF APP0 segment
- JFIF extension (JFXX) APP0 segment (read only)
- EXIF APP1 segment
- Photoshop Image Resource APP13 marker segment(it includes some IPTC data and another Photoshop Image Resource metadata structure)
- Other application marker segment levels

JFIF APP0 segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
“APP0”	0xFFE0	LEVEL_START	0	NULL	0	TRUE
“APP0_DATASIZE”	60224	VALUE_ITEM	UINT16		1	FALSE
“JFIF_HEADER”	200	VALUE_ITEM	STRING	JFIF	5	TRUE
“Version”	201	VALUE_ITEM	UINT16		1	FALSE
“ResolutionUnits”	202	VALUE_ITEM	UINT8		1	FALSE
“ResolutionX”	203	VALUE_ITEM	UINT16		1	FALSE
“ResolutionY”	204	VALUE_ITEM	UINT16		1	FALSE
“ThumbnailWidth”[a]	205	VALUE_ITEM	UINT8		1	TRUE
“ThumbnailHeight”[b]	206	VALUE_ITEM	UINT8		1	TRUE
“APP0”	0xFFE0	LEVEL_END	0	NULL	0	TRUE

a.The thumbnail width tags are present only if the thumbnail is stored to JFIF segment.

b.The thumbnail height tags are present only if the thumbnail is stored to JFIF segment.

JFIF extension (JFXX) APP0 segment

Name	Id	Type	Value Type	Value	Value Length	Read Only

“APP0”	0xFFE0	LEVEL_START	0	NULL	0	TRUE
“APP0_DATASIZE”	60224	VALUE_ITEM	UINT16		1	TRUE
“JFIF_EX_HEADER”	209	VALUE_ITEM	STRING	“JFXX”	5	TRUE
“ExtensionCode”	210	VALUE_ITEM	UINT8		1	TRUE
“APP0”	0xFFE0	LEVEL_END	0	NULL	0	TRUE

Photoshop Image Resource APP13 marker segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
“APP13”	0xFFED	LEVEL_START	0	NULL	0	TRUE
“APP13_DATASIZE”	60237	VALUE_ITEM	UINT16		1	FALSE
“PHOTOSHOP_HEADER”	59915	VALUE_ITEM	STRING	“Photoshop 3.0”	14	FALSE
Photoshop Image Resource level 1						
...						
Photoshop Image Resource level n						
“APP13”	0xFFED	LEVEL_END	0	NULL	0	TRUE

NOTE: For **“Photoshop Image Resource”** metadata structure see in TIFF non-image data Photoshop Image Resource metadata structure Table.

Other application marker segment levels

Name	Id	Type	Value Type	Value	Value Length	Read Only
<Marker name>	<Marker Id>	LEVEL_START	0	NULL	0	TRUE
<Marker size tag name>	<Marker size tag Id>	VALUE_ITEM	UINT16		1	FALSE

“DATA”	<Marker Id>	VALUE_ITEM	RAW_DATA		<ValueSize>	FALSE
<Marker name>	<Marker Id>	LEVEL_END	0	NULL	0	TRUE

<Marker Id>, **<Marker name>**, **<Marker size tag name>** and **<Marker size tag Id>** are in table below:

Marker	<Marker name>	<Marker Id>	<Marker size tag name>	<Marker size tag Id>
APP0	“APP0”	0xFFE0	“APP0_DATASIZE”	60224
APP1	“APP1”	0xFFE1	“APP1_DATASIZE”	60225
APP2	“APP2”	0xFFE2	“APP2_DATASIZE”	60226
APP3	“APP3”	0xFFE3	“APP3_DATASIZE”	60227
APP4	“APP4”	0xFFE4	“APP4_DATASIZE”	60228
APP5	“APP5”	0xFFE5	“APP5_DATASIZE”	60229
APP6	“APP6”	0xFFE6	“APP6_DATASIZE”	60230
APP7	“APP7”	0xFFE7	“APP7_DATASIZE”	60231
APP8	“APP8”	0xFFE8	“APP8_DATASIZE”	60232
APP9	“APP9”	0xFFE9	“APP9_DATASIZE”	60233
APP10	“APP10”	0xFFEA	“APP10_DATASIZE”	60234
APP11	“APP11”	0xFFEB	“APP11_DATASIZE”	60235
APP12	“APP12”	0xFFEC	“APP12_DATASIZE”	60236
APP13	“APP13”	0xFFED	“APP13_DATASIZE”	60237
APP14	“APP14”	0xFFEE	“APP14_DATASIZE”	60238
APP15	“APP15”	0xFFEF	“APP15_DATASIZE”	60239

Callback required for writing JPEG metadata items

All JPEG metadata items can be written using **LPAFT_IG_METAD_ITEM_ADD_CB** callback only.

NOTE: Please see file `metadata.h` for the full list of JPEG Metadata **Item Name** and **Item Id** constants.

EXIF-TIFF Non-Image Data Structure

EXIF-TIFF metadata structure is framed in several levels –

- EXIF-TIFF level containing all EXIF-TIFF data
- IFD0 level
- Tag levels

EXIF-TIFF level

The EXIF-TIFF metadata structure is similar to TIFF's one. But EXIF-TIFF allows Exif subIFD tags, GPS subIFD tags and thumbnail IFD tags to be parsed and passed together with main IFD.

Name	Id	Type	Value Type	Value	Value Length	Read Only
"EXIF"	`IG_FORMAT_EXIF_TIFF`	`LEVEL_START`	0	`NULL`	0	`TRUE`
"TIF_HEADER"	59936	`VALUE_ITEM`	UINT16	0x4949 or 0x4D4D	1	`TRUE`
IFD 0 level (if present)						
Thumbnail IFD level (if present)						
"EXIF"	`IG_FORMAT_EXIF_TIFF`	`LEVEL_END`	0	`NULL`	0	`TRUE`

All items between items with Name "EXIF" and Id `IG_FORMAT_EXIF_TIFF` (Type `LEVEL_START` and `LEVEL_END`) are interpreted as EXIF data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

See also Exif subIFD tags for the list of Exif subIFD tag names and Ids.

IFD0 level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IFD”	0	LEVEL_START	0	NULL	0	TRUE
Exif subIFD, GPS info subIFD and tags levels mixed in any way						
“IFD”	0	LEVEL_END	0	NULL	0	TRUE

Exif subIFD level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IFD”	34665	LEVEL_START	0	NULL	0	TRUE
Interoperability info subIFD and tags levels mixed in any way						
“IFD”	34665	LEVEL_END	0	NULL	0	TRUE

The number 34665 is the Exif subIFD pointer tag (see below)

IFD1 (Thumbnail IFD), GPS info subIFD and Interoperability info subIFD levels

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IFD”	<IFDId>	LEVEL_START	0	NULL	0	TRUE
Tag 1						
...						
Tag n						
“IFD”	<IFDId>	LEVEL_END	0	NULL	0	TRUE

The value of <IFDId> is 1 for IFD1 or the subIFD pointer tag identifier for GPS (34853)(see GPS subIFD tags) or Interoperability (40965) subIFD (see Interoperability subIFD tags).

Tag levels

For tag metadata structure see TIFF Non-Image Data Structure.

Description of tags used in EXIF

There are such types of EXIF tags described:

- Exif subIFD tags
- GPS subIFD tags
- Interoperability subIFD tags

For IFD0 and Thumbnail IFD tags see TIFF Non-Image Data Structure.

Exif subIFD tags

Item Name	Item Id	Read Only
“ExposureTime”	33434	FALSE
“Fnumber”	33437	FALSE
“ExposureProgram”	34850	FALSE
“SpectralSensitivity”	34852	FALSE
“ISOSpeedRatings”	34855	FALSE
“OECF”	34856	FALSE
“ExifVersion”	36864	FALSE
“DateTimeOriginal”	36867	FALSE
“DateTimeDigitized”	36868	FALSE
“ComponentsConfiguration”	37121	FALSE
“CompressedBitsPerPixel”	37122	FALSE
“ShutterSpeedValue”	37377	FALSE
“ApertureValue”	37378	FALSE
“BrightnessValue”	37379	FALSE
“ExposureBiasValue”	37380	FALSE
“MaxApertureValue”	37381	FALSE
“SubjectDistance”	37382	FALSE
“MeteringMode”	37383	FALSE

“LightSource”	37384	FALSE
“Flash”	37385	FALSE
“FocalLength”	37386	FALSE
"SubjectArea"	37396	FALSE
“MakerNote”	37500	FALSE
“UserComment”	37510	FALSE
“SubSecTime”	37520	FALSE
“SubSecTimeOriginal”	37521	FALSE
“SubSecTimeDigitized”	37522	FALSE
“FlashPixVersion”	40960	FALSE
“ColorSpace”	40961	FALSE
“PixelXDimension”	40962	TRUE
“PixelYDimension”	40963	TRUE
“RelatedSoundFile”	40964	FALSE
“InteroperabilityIFDPointer”	40965	TRUE
“FlashEnergy”	41483	FALSE
“SpatialFrequencyResponse”	41484	FALSE
“FocalPlaneXResolution”	41486	FALSE
“FocalPlaneYResolution”	41487	FALSE
“FocalPlaneResolutionUnit”	41488	FALSE
“SubjectLocation”	41492	FALSE
“ExposureIndex”	41493	FALSE
“SensingMethod”	41495	FALSE
“FileSource”	41728	FALSE
“SceneType”	41729	FALSE
“CFAPattern”	41730	FALSE
"CustomRendered"	41985	FALSE
"ExposureMode"	41986	FALSE
"WhiteBalance"	41987	FALSE
"DigitalZoomRatio"	41988	FALSE

"FocalLengthIn35mmFilm"	41989	FALSE
"SceneCaptureType"	41990	FALSE
"GainControl"	41991	FALSE
"Contrast"	41992	FALSE
"Saturation"	41993	FALSE
"Sharpness"	41994	FALSE
"DeviceSettingDescription"	41995	FALSE
"SubjectDistanceRange"	41996	FALSE
"ImageUniqueID"	42016	FALSE

GPS subIFD tags

Item Name	Item Id	Read Only
“GPSVersionID”	0	FALSE
“GPSLatitudeRef”	1	FALSE
“GPSLatitude”	2	FALSE
“GPSLongitudeRef”	3	FALSE
“GPSLongitude”	4	FALSE
“GPSAltitudeRef”	5	FALSE
“GPSAltitude”	6	FALSE
“GPSTimeStamp”	7	FALSE
“GPSSatellites”	8	FALSE
“GPSStatus”	9	FALSE
“GPSMeasureMode”	10	FALSE
“GPSDOP”	11	FALSE
“GPSSpeedRef”	12	FALSE
“GPSSpeed”	13	FALSE
“GPSTrackRef”	14	FALSE
“GPSTrack”	15	FALSE

”GPSImgDirectionRef”	16	FALSE
“GPSImgDirection”	17	FALSE
“GPSMapDatum”	18	FALSE
“GPSDestLatitudeRef”	19	FALSE
“GPSDestLatitude”	20	FALSE
“GPSDestLongitudeRef”	21	FALSE
“GPSDestLongitude”	22	FALSE
“GPSDestBearingRef”	23	FALSE
“GPSDestBearing”	24	FALSE
“GPSDestDistanceRef”	25	FALSE
“GPSDestDistance”	26	FALSE
"GPSProcessingMethod"	27	FALSE
"GPSAreaInformation"	28	FALSE
"GPSDateStamp"	29	FALSE
"GPSDifferential"	30	FALSE

Interoperability subIFD tags

Item Name	Item Id	Read Only
“InteroperabilityIndex “	1	FALSE
“InteroperabilityVersion “	2	FALSE
“RelatedImageFileFormat”	4096	FALSE
“RelatedImageWidth “	4097	FALSE
“RelatedImageLength”	4098	FALSE

Callback required for writing EXIF metadata items.

Value of these tags can be changed using **`LPAFT_IG_METAD_ITEM_SET_CB`** callback only.

Item Name	Item Id
“TIF_HEADER”	59936

IFD0 tags	
“PlanarConfiguration”	284
“YCbCrSubSampling”	530
“RowsPerStrip”	278
“YCbCrPositioning”	531
“XResolution”	282
“YResolution”	283
“ResolutionUnit”	296
Exif subIFD	
“ExifVersion”	36864
“FlashPixVersion”	40960
“ColorSpace”	40961
Thumbnail IFD	
“ImageWidth”	256
“ImageLength”	257
“PlanarConfiguration”	284
“YCbCrSubSampling”	530
“RowsPerStrip”	278
“YCbCrPositioning”	531
“XResolution”	282
“YResolution”	283
“ResolutionUnit”	296

The rest of metadata item values can be written using **`LPAFT_IG_METAD_ITEM_ADD_CB`** callback only.

EXIF-TIFF Metadata Item Name and ID constants

NOTE: Please see file `metadata.h` for the full list of EXIF Metadata **Item Name** and **Item Id** constants.

EXIF-JPEG Non-image Data Structure

The EXIF-JPEG metadata structure is similar to JPEG one. But EXIF-JPEG sends the Exif APP1 marker segment data in parsed form and Jfif APP0 marker segment data is not parsed and sent as raw data.

EXIF-JPEG level

All items between items with Name “EXIF” and Id `IG_FORMAT_EXIF_JPEG` (Type `LEVEL_START` and `LEVEL_END`) are interpreted as EXIF data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

For JPEG marker segment levels see JPEG Non-Image Data Structure.

Name	Id	Type	Value Type	Value	Value Length	Read Only
“EXIF-JPEG”	`IG_FORMAT_EXIF_JPEG`	`LEVEL_START`	0	`NULL`	0	TRUE
Exif APP1 marker segment level						
JPEG marker segment levels mixed in any way (if present)						
“EXIF-JPEG”	`IG_FORMAT_EXIF_JPEG`	`LEVEL_END`	0	`NULL`	0	TRUE

Exif APP1 marker segment level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“APP1”	65505	`LEVEL_START`	0	`NULL`	0	TRUE
“APP1_DATASIZE”	60225	`VALUE_ITEM`	UINT16	<Data>	1	TRUE
“EXIF_HEADER”	59935	`VALUE_ITEM`	STRING	“EXIF”	5	TRUE
“TIF_HEADER”	59936	`VALUE_ITEM`	UINT16	0x4949 or 0x4D4D	1	TRUE
IFD 0 level (if present)						

Thumbnail IFD level (if present)						
“APP1”	0xFFE1	LEVEL_END	0	NULL	0	TRUE

NOTE: The “APP1_DATASIZE” item can be omitted during saving.

For more detailed Exif information see EXIF-TIFF Non-Image Data Structure.

EXIF Makernote

Makernote is a standard EXIF tag of UNDEFINED (byte) type. This tag usually is used as a "hidden" IFD. Makernote can not be read in and written out as a BLOB, because IFD offsets become invalid. Therefore, to preserve this tag while writing an image it's necessary to decode it during reading and re-encode it during writing, despite the fact that formally makernote is just a binary tag.

There are several types of makernotes, all of them have corresponding constants in Metadata.h file:

IG_MAKERNOTE_TYPE_UNKNOWN	This is the default type. This type means that we can’t decect Makernote as any other type. Preserving such makernote and saving it with the file, most likely, does not make any sence, because IFD offsets will be corrupted.
IG_MAKERNOTE_TYPE_IFD	Makernote is a valid TIF IFD, except that it is not referenced from anywhere in the file, and does not reference other IFDs. Such makernote is preserved entirely when writing to a file.
IG_MAKERNOTE_TYPE_IFD_PREFIXED	Same as IG_MAKERNOTE_TYPE_IFD, but with a short prefix before the IFD. The prefix is also preserved, so the whole Makernote*s* is preserved when writing to a file.
IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED	Makernote starts with a prefix, then goes TIF image header, which points to an IFD.
IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II	Makernote starts with a prefix, then goes offset to the IFD, then IFD itself. Makernote IFD uses Intel byte ordering (II), even though the whole file uses Motorola ordering.

To present Makernote info in a convenient way ImageGear introduces a special structure as an addition to the standard EXIF Metadata: "Makernote*s* Wrapper IFD". Instead of a single binary Makernote*s* tag (37500), we are adding a virtual IFD, containing all the information about Makernote*s*.

'Makernote Wrapper IFD' has the following format depending on the 'type' tag:

a. Type `IG_MAKERNOTE_TYPE_UNKNOWN`:

```
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_UNKNOWN
-------->'binary MakerNote'
```

b. Type `IG_MAKERNOTE_TYPE_IFD`:

```
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_IFD
-------->'MakerNote IFD'
----------------> 'individual MakerNote tag 1'
----------------> 'individual MakerNote tag 2'
...
----------------> 'individual MakerNote tag N'
-------->'binary MakerNote'
```

c. Type `IG_MAKERNOTE_TYPE_IFD_PREFIXED`:

```
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_IFD_PREFIXED
-------->'MakerNote IFD prefix'
-------->'MakerNote IFD'
----------------> 'individual MakerNote tag 1'
----------------> 'individual MakerNote tag 2'
...
----------------> 'individual MakerNote tag N'
-------->'binary MakerNote'
```

d. Type `IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED`:

```
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED
<The rest of the structure is identical to
IG_MAKERNOTE_TYPE_IFD_PREFIXED>
```

e. Type `IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II`:

```
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II
<The rest of the structure is identical to
IG_MAKERNOTE_TYPE_IFD_PREFIXED>
```

If 'Makernote' EXIF tag is present in the file then ImageGear will always provide it in its original binary form via meta-data callback ('binary Makernote' tag) on the read side, and will also optionally provide it as a decoded sub-IFD (w/ or w/o prefix depending on how it is stored in the original file) if it can be decoded.

ImageGear will ignore 'binary Makernote' tag on the write side for the 'Makernote wrapper IFDs' that have 'type' other than `IG_MAKERNOTE_TYPE_UNKNOWN` - and construct 'Makernote' EXIF tag based on the 'Makernote IFD'.

ImageGear will store 'binary Makernote' to file 'as is' for the 'Makernote wrapper IFDs' that have 'type' set to `IG_MAKERNOTE_TYPE_UNKNOWN`.

If 'Makernote wrapper IFD' is not provided to ImageGear during saving operation then 'Makernote' EXIF tag will not be saved into the output file.

Makernote IFD tags are listed below:

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IFD”	***)	LEVEL_START	0	NULL		FALSE
“MakerNoteType”		VALUE_ITEM	UINT16	*)	1	FALSE
“MakerNotePrefix”		VALUE_ITEM	RAW_DATA			FALSE
“MakerNoteBinary”		VALUE_ITEM	RAW_DATA			FALSE
“IFD”		LEVEL_START	0	NULL	0	FALSE
“UNDEFINED”** *)		VALUE_ITEM	RAW_DATA			FALSE
...						
“IFD”		LEVEL_END	0	NULL	0	FALSE
“IFD”		LEVEL_END	0	NULL		FALSE

*) One of the following `enumIGEXIFMakerNoteType` constants:

```
IG_MAKERNOTE_TYPE_UNKNOWN = 0
IG_MAKERNOTE_TYPE_IFD = 1
IG_MAKERNOTE_TYPE_IFD_PREFIXED = 2
IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED = 3
IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II = 4
```

**) Manufacturer specific tags, not covered by EXIF standard.

***) See respective `IGMDTAG_ID_EXIF_MAKERNOTE_` Id constants in the section Custom Makernote Tags and IFDs

Custom Makernote Tags and IFDs

The following custom tags and IFDs are introduced for Makernote support:

Makernote Wrapper IFD: tag = 37500 (same as standard Makernote tag). This IFD is located in the “Exif IFD” (34665).

New tags, which belong to the Makernote Wrapper IFD:

```
enum enumIGEXIFMakerNoteTagIDs
{
IGMDTAG_ID_EXIF_MAKERNOTE_TYPE = 1,
IGMDTAG_ID_EXIF_MAKERNOTE_PREFIX = 2,
IGMDTAG_ID_EXIF_MAKERNOTE_BINARY = 1000,
IGMDTAG_ID_EXIF_MAKERNOTE_DATA_IFD = 1001
```

```
};
```

Vendors and models currently supported

Make	Model
Canon	Canon DIGITAL IXUS
Canon	Canon EOS D30
Canon	Canon PowerShot G2
Canon	Canon PowerShot S50
FUJIFILM	FinePix4900Z
LEICA	digilux 4.3
Minolta Co., Ltd.	DiMAGE 7i
Nikon	E5000 (TIF)
NIKON	E5700
NIKON	E950
OLYMPUS OPTICAL CO.,LTD	C2040Z
OLYMPUS OPTICAL CO.,LTD	C960Z,D460Z
Panasonic	DMC-LC5
RICOH	Caplio RR1
SANYO Electric Co.,Ltd.	SR6
SANYO Electric Co.,Ltd.	SX113
SANYO Electric Co.,Ltd.	SX212
SEIKO EPSON CORP.	PhotoPC 850Z

Exif APP2 marker segment (Flashpix extensions) levels

EXIF file format allows to store Flashpix extensions in APP2 marker segments. There are several types of FPXR APP2 marker segments described below:

- FPXR Contents List APP2 marker segment
- Interoperability Entity level
- FPXR Data Stream APP2 marker segment
- The other FPXR APP2 marker segment

FPXR Contents List APP2 marker segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
“APP2”	0xFFE2	LEVEL_START	0	NULL	0	TRUE
“APP2_DATASIZE”	60226	VALUE_ITEM	UINT16		1	TRUE
“FPXR_HEADER”	61221	VALUE_ITEM	STRING	“FPXR”	4	FALSE
“FPXRVersion”	61222	VALUE_ITEM	UINT8		1	FALSE
“FPXRExtensionID”	61223	VALUE_ITEM	UINT8	“1”	1	FALSE
“FPXRInteroperabilityCount”	61224	VALUE_ITEM	UINT16	<n>	1	FALSE
Interoperability Entity level 0						
...						
Interoperability Entity level <n> - 1						
“APP2”	0xFFE2	LEVEL_END	0	NULL	0	TRUE

Interoperability Entity level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“InteroperabilityEntity”	<Ind.>	LEVEL_START	0	NULL	0	TRUE
“EntitySize”	<Ind.>	VALUE_ITEM	UINT32	0xFFFFFFFF for Storage or <Variable> for Stream	1	FALSE
“DefaultValue”	<Ind.>	VALUE_ITEM	UINT8	“FPXR”	4	FALSE
“Storage/StreamName”	<Ind.>	VALUE_ITEM	RAW_DATA	<Unicode name>	<Variable>	FALSE
The next item is present only if “EntitySize” item value is equal 0xFFFFFFFF (Storage)						

“EntityClassID”	<Ind.>	VALUE_ITEM	RAW_DATA		16	FALSE
“InteroperabilityEntity”	<Ind.>	LEVEL_END	0	NULL	0	TRUE

FPXR Data Stream APP2 marker segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
“APP2”	0xFFE2	LEVEL_START	0	NULL	0	TRUE
“APP2_DATASIZE”	60226	VALUE_ITEM	UINT16		1	TRUE
“FPXR_HEADER”	61221	VALUE_ITEM	STRING	“FPXR”	4	FALSE
“FPXRVersion”	61222	VALUE_ITEM	UINT8		1	FALSE
“FPXRExtensionID”	61223	VALUE_ITEM	UINT8	“2”	1	FALSE
“FPXRIndexToContentsList”	61225	VALUE_ITEM	UINT16	<Ind.>	1	FALSE
“FPXROffsetToStream”	61226	VALUE_ITEM	UINT32		1	FALSE
“FPXRStreamData”	61227	VALUE_ITEM	RAW_DATA		<Variable>	FALSE
“APP2”	0xFFE2	LEVEL_END	0	NULL	0	TRUE

<Ind.> is pointer to appropriate “InteroperabilityEntity” of FPXR Contents List APP2 marker segment. <Ind.> value is between 0 and <n> - 1(<n> is the “FPXRInteroperabilityCount” item value of FPXR Contents List APP2 marker segment).

The other FPXR APP2 marker segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
“APP2”	0xFFE2	LEVEL_START	0	NULL	0	TRUE
“APP2_DATASIZE”	60226	VALUE_ITEM	UINT16		1	TRUE
“FPXR_HEADER”	61221	VALUE_ITEM	STRING	“FPXR”	4	FALSE

“FPXRVersion”	61222	VALUE_ITEM	UINT8		1	FALSE
“FPXRExtensionID”	61223	VALUE_ITEM	UINT8	More than “2”	1	FALSE
“FPXRData”	61228	VALUE_ITEM	RAW_DATA		<Variable>	FALSE
“APP2”	0xFFE2	LEVEL_END	0	NULL	0	TRUE

EXIF-JPEG and EXIF-FPX Metadata Item Name and ID constants

NOTE: Please see file `metadata.h` for the full list of EXIF-JPEG and EXIF APP2 Marker segment Metadata **Item Name** and **Item Id** constants.

CIFF Non-Image Data Structure

CIFF metadata is similar to EXIF-TIFF, but it uses multiple nested IFDs, structured as a tree.

Brief information on CIFF metadata levels is provided in the set of tables below.

NOTE: Please see file `metadata.h` for the full list of CIFF Metadata **Item Name** and **Item Id** constants.

CIFF level

Name	Id	Type
“CIFF”	IG_FORMAT_CIFF	LEVEL_START
ImageProps level		
CIFF root tags		
“CIFF”	IG_FORMAT_CIFF	LEVEL_END

ImageProps level

Name	Id	Type
“ImageProps”	IGMDTAG_ID_CIFF_IMAGE_PROPS	LEVEL_START
ImageInfo level		
ImageDescription level		
CameraObject level		
ShootingRecord level		
MeasuredInfo level		
ExifInformation level		
ImageProps tags		
“ImageProps”	IGMDTAG_ID_CIFF_IMAGE_PROPS	LEVEL_END

ImageInfo level

Name	Id	Type
“ImageInfo”	IGMDTAG_ID_CIFF_IMAGE_INFO	LEVEL_START
ImageInfo tags		
“ImageInfo”	IGMDTAG_ID_CIFF_IMAGE_INFO	LEVEL_END

ImageDescription level

Name	Id	Type
“ImageDescription”	IGMDTAG_ID_CIFF_IMAGE_DESCRIPTION	LEVEL_START
ImageDescription tags		
“ImageDescription”	IGMDTAG_ID_CIFF_IMAGE_DESCRIPTION	LEVEL_END

CameraObject level

Name	Id	Type
“CameraObject”	IGMDTAG_ID_CIFF_CAMERA_OBJECT	LEVEL_START
CameraSpecification level		
CameraObject tags		
“CameraObject”	IGMDTAG_ID_CIFF_CAMERA_OBJECT	LEVEL_END

ShootingRecord level

Name	Id	Type
“ShootingRecord”	IGMDTAG_ID_CIFF_SHOOTING_RECORD	LEVEL_START
ShootingRecord tags		
“ShootingRecord”	IGMDTAG_ID_CIFF_SHOOTING_RECORD	LEVEL_END

MeasuredInfo level

Name	Id	Type
“MeasuredInfo”	IGMDTAG_ID_CIFF_MEASURED_INFO	LEVEL_START
MeasuredInfo tags		
“MeasuredInfo”	IGMDTAG_ID_CIFF_MEASURED_INFO	LEVEL_END

ExifInformation level

Name	Id	Type

“ExifInformation”	IGMDTAG_ID_CIFF_EXIF_INFORMATION	LEVEL_START
FocalLength level		
CanonShotInfo level		
CanonCameraSettings level		
SensorInfo level		
CanonCustomFunctions level		
CanonPictureInfo level		
ExifInformation tags		
“ExifInformation”	IGMDTAG_ID_CIFF_EXIF_INFORMATION	LEVEL_END

CameraSpecification level

Name	Id	Type
“CameraSpecification”	IGMDTAG_ID_CIFF_CAMERA_SPECIFICATION	LEVEL_START
CameraSpecification tags		
“CameraSpecification”	IGMDTAG_ID_CIFF_CAMERA_SPECIFICATION	LEVEL_END

FocalLength level

Name	Id	Type
“FocalLength”	IGMDTAG_NAME_CIFF_FOCAL_LENGTH	LEVEL_START
FocalLength tags		
“FocalLength”	IGMDTAG_NAME_CIFF_FOCAL_LENGTH	LEVEL_END

CanonShotInfo level

Name	Id	Type
“CanonShotInfo”	IGMDTAG_NAME_CIFF_CANON_SHOT_INFO	LEVEL_START
CanonShotInfo tags		
“CanonShotInfo”	IGMDTAG_NAME_CIFF_CANON_SHOT_INFO	LEVEL_END

CanonCameraSettings level

Name	Id	Type
“CanonCameraSettings”	IGMDTAG_NAME_CIFF_CANON_CAMERA_SETTINGS	LEVEL_START
CanonCameraSettings tags		
“CanonCameraSettings”	IGMDTAG_NAME_CIFF_CANON_CAMERA_SETTINGS	LEVEL_END

SensorInfo level

Name	Id	Type
“SensorInfo”	IGMDTAG_NAME_CIFF_SENSOR_INFO	LEVEL_START
SensorInfo tags		
“SensorInfo”	IGMDTAG_NAME_CIFF_SENSOR_INFO	LEVEL_END

CanonCustomFunctions level

Name	Id	Type
“CanonCustomFunctions”	IGMDTAG_NAME_CIFF_CANON_CUSTOM_FUNCTIONS	LEVEL_START
CanonCustomFunctions tags		
“CanonCustomFunctions”	IGMDTAG_NAME_CIFF_CANON_CUSTOM_FUNCTIONS	LEVEL_END

CanonPictureInfo level

Name	Id	Type
“CanonPictureInfo”	IGMDTAG_NAME_CIFF_CANON_PICTURE_INFO	LEVEL_START
CanonPictureInfo tags		
“CanonPictureInfo”	IGMDTAG_NAME_CIFF_CANON_PICTURE_INFO	LEVEL_END

IPTC Non-Image Data Structure

IPTC, International Press and Telecommunications Council Standards have been created for different type information exchanging.

IPTC is not file format. It can be stored within the JPEG, TIFF, EXIF-JPEG and EXIF-TIFF image file as Adobe Photoshop Image Resource (see Photoshop Image Resource metadata structure). In TIFF and EXIF-TIFF files the IPTC data is stored in TIFF tag with Id = 34377 (see TIFF Non-Image Data Structure for metadata structure of the tag). In JPEG and EXIF-JPEG files the IPTC data is stored in Photoshop APP13 marker segment (see JPEG Non-Image Data Structure).

The ImageGear IPTC metadata structure is framed in two levels:

- IPTC level containing all IPTC data
- Record level containing value items of a record.

IPTC level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IPTC”	0x1C00	LEVEL_START	UNDEFINED	NULL	0	TRUE
Record level 1						
...						
Record level n						
“IPTC”	0x1C00	LEVEL_END	UNDEFINED	NULL	0	TRUE

All items between items with Name “IPTC” and Id 0x1C00 (Type **LEVEL_START** and **LEVEL_END**) are interpreted as IPTC data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

Record level

Name	Id	Type	ValueType	Value	Value Length	Read Only
“IPTC_RECORD”	<Record#>	LEVEL_START	UNDEFINED	NULL	0	TRUE
Dataset value item						
…						
Dataset value item						
“IPTC_RECORD”	<Record#>	LEVEL_END	UNDEFINED	NULL	0	TRUE

The **<Record#>** is identifier of IPTC record. Its value is the number of appropriate IPTC record described in IPTC – NAA IIM4 specification. Record levels must follow in numerical order within IPTC level.

Dataset value item

Name	Id	Type	ValueType	Value	Value Length	Read Only
<Dataset name>	<Dataset Id>	VALUE_ITEM	<Value Type>	<Data>	<Value Length>	FALSE

The available **<Dataset name>**, **<Dataset Id>**, **<Value Length>** and **<Value Type>** values are described below for each IPTC record according to IPTC-NAA IIM4 specification. Dataset value item can follow in any order within appropriate record level.

Dataset value items description for IPTC Envelope Record (Record #1)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
0	"ModelVersion"	AT_TID_WORD	1
5	"Destination"	AT_TID_STRING	1024
20	"FileFormat"	AT_TID_WORD	1
22	"FileFormatVersion"	AT_TID_WORD	1
30	"ServiceIdentifier"	AT_TID_STRING	10
40	"EnvelopeNumber"	AT_TID_STRING	8
50	"ProductID"	AT_TID_STRING	32
60	"EnvelopePriority"	AT_TID_STRING	1
70	"DateSent"	AT_TID_STRING	8
80	"TimeSent"	AT_TID_STRING	11
90	"CodedCharacterSet"	AT_TID_STRING	32
100	"UNO"	AT_TID_STRING	80
120	"ARMIdentifier"	AT_TID_WORD	1
122	"ARMVersion"	AT_TID_WORD	1

Dataset value items description for IPTC Application Record (Record #2)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
0	"RecordVersion"	AT_TID_WORD	1
3	"ObjectTypeReference"	AT_TID_STRING	67
4	"ObjectAttributeReference"	AT_TID_STRING	68
5	"ObjectName"	AT_TID_STRING	64
7	"EditStatus"	AT_TID_STRING	64
8	"EditorialUpdate"	AT_TID_STRING	2
10	"Urgency"	AT_TID_STRING	1
12	"SubjectReference"	AT_TID_STRING	236
15	"Category"	AT_TID_STRING	3
20	"SupplementalCategory"	AT_TID_STRING	32
22	"FixtureIdentifier"	AT_TID_STRING	32
25	"Keywords"	AT_TID_STRING	64
26	"ContentLocationCode"	AT_TID_STRING	3
27	"ContentLocationName"	AT_TID_STRING	64
30	"ReleaseDate"	AT_TID_STRING	8
35	"ReleaseTime"	AT_TID_STRING	11
37	"ExpirationDate"	AT_TID_STRING	8
38	"ExpirationTime"	AT_TID_STRING	11
40	"SpecialInstructions"	AT_TID_STRING	256
42	"ActionAdvised"	AT_TID_STRING	2
45	"ReferenceService"	AT_TID_STRING	10
47	"ReferenceDate"	AT_TID_STRING	8
50	"ReferenceNumber"	AT_TID_STRING	8
55	"DateCreated"	AT_TID_STRING	8

60	"TimeCreated"	AT_TID_STRING	11
62	"DigitalCreationDate"	AT_TID_STRING	8
63	"DigitalCreationTime"	AT_TID_STRING	11
65	"OriginatingProgram"	AT_TID_STRING	32
70	"ProgramVersion"	AT_TID_STRING	10
75	"ObjectCycle"	AT_TID_STRING	1
80	"By-line"	AT_TID_STRING	32
85	"By-lineTitle"	AT_TID_STRING	32
90	"City"	AT_TID_STRING	32
92	"Sublocation"	AT_TID_STRING	32
95	"Province/State"	AT_TID_STRING	32
100	"Country/ PrimaryLocationCode"	AT_TID_STRING	3
101	"Country/ PrimaryLocationName"	AT_TID_STRING	64
103	"OriginalTransmission Reference"	AT_TID_STRING	32
105	“Headline"	AT_TID_STRING	256
110	"Credit"	AT_TID_STRING	32
115	"Source"	AT_TID_STRING	32
116	"CopyrightNotice"	AT_TID_STRING	128
118	"Contact"	AT_TID_STRING	128
120	"Caption/Abstract"	AT_TID_STRING	2000
122	"Writer/Editor"	AT_TID_STRING	32
125	"RasterizedCaption"	AT_TID_BYTE	7360
130	"ImageType"	AT_TID_STRING	2
131	"ImageOrientation"	AT_TID_STRING	1
135	"LanguageIdentifier"	AT_TID_STRING	3
150	"AudioType"	AT_TID_STRING	2
151	"AudioSamplingRate"	AT_TID_STRING	6
152	"AudioSamplingResolution"	AT_TID_STRING	2

153	"AudioDuration"	AT_TID_STRING	6
154	"AudioOutcue"	AT_TID_STRING	64
200	"ObjectDataPreview FileFormat"	AT_TID_WORD	1
201	"ObjectDataPreview FileFormatVersion"	AT_TID_WORD	1
202	"ObjectDataPreviewData"	AT_TID_BYTE	Undefined

Dataset value items description for IPTC Digital News Photo Parameter Record (Record #3)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
0	"RecordVersion"	AM_TID_TXT_UINT16	1
10	"PictureNumber"	AM_TID_RAW_DATA	16
20	"PixelsPerLine"	AM_TID_TXT_UINT16	1
30	"NumberOfLine"	AM_TID_TXT_UINT16	1
40	"PixelSizeInScanningDirection"	AM_TID_TXT_UINT16	1
50	"PixelSizePerpendicularTo ScanningDirection"	AM_TID_TXT_UINT16	1
55	"SupplementType"	AM_TID_TXT_UINT8	1
60	"ColourRepresentation"	AM_TID_TXT_UINT8	2
64	"InterchangeColourSpace"	AM_TID_TXT_UINT8	1
65	"ColourSequence"	AM_TID_TXT_UINT8	4
66	"ICCInputColourProfile"	AM_TID_RAW_DATA	Undefined
70	"ColourCalibrationMatrixTable"	AM_TID_RAW_DATA	Undefined
80	"LookupTable"	AM_TID_RAW_DATA	131072

84	"NumberOfIndexEntries"	AM_TID_TXT_UINT16	1
85	"ColourPalette"	AM_TID_RAW_DATA	Undefined
86	"NumberOfBitsPerSample"	AM_TID_TXT_UINT8	1
90	"SamplingStructure"	AM_TID_TXT_UINT8	1
100	"ScanningDirection"	AM_TID_TXT_UINT8	1
102	"ImageRotation"	AM_TID_TXT_UINT8	1
110	"DataCompressionMethod"	AM_TID_RAW_DATA	4
120	"QuantisationMethod"	AM_TID_TXT_UINT8	1
125	"EndPoints"	AM_TID_TXT_UINT8	Undefined
130	"ExcursionTolerance"	AM_TID_TXT_UINT8	1
135	"BitsPerComponent"	AM_TID_TXT_UINT8	Undefined
140	"MaximumDensityRange"	AM_TID_TXT_UINT16	1
145	"GammaCompensatedValue"	AM_TID_TXT_UINT16	1

Dataset value items description for IPTC Pre-Object Descriptor Record (Record #7

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
10	"SizeMode"	AM_TID_TXT_UINT8	1
20	"MaxSubfileSize"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1
90	"ObjectDataSizeAnnounced"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1

95	"MaximumObjectDataSize"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1

Dataset value items description for IPTC Object Record (Record #8)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
10	“Subfile”	AM_TID_RAW_DATA	Undefined

Dataset value items description for IPTC Post-Object Descriptor Record (Record #9)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
10	"ConfirmedObjectDataSize"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1

Callback required for writing IPTC metadata items

All IPTC metadata items can be written using **LPAFT_IG_METAD_ITEM_ADD_CB** callback only.

IPTC Metadata Item Name and ID constants

NOTE: Please see file `metadata.h` for the full list of IPTC Metadata **Item Name** and **Item Id** constants.

PNG Non-Image Data Structure

There are 2 main PNG metadata levels:

- PNG metadata level
- PNG chunks levels

PNG metadata level

All items between items with Name “PNG” and Id `IG_FORMAT_PNG` (Type `LEVEL_START` and `LEVEL_END`) are interpreted as PNG data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

Name	Id	Type	Value Type	Value	Value Length	Read Only
“PNG”	`IG_FORMAT_PNG`	`LEVEL_START`	0	`NULL`	0	`TRUE`
PNG chunks levels						
“PNG”	`IG_FORMAT_PNG`	`LEVEL_END`	0	`NULL`	0	`TRUE`

PNG chunks levels

There are such PNG non-image chunk levels supported by ImageGear:

- PNG header chunk metadata
- Physical dimension chunk metadata
- Transparency chunk metadata
- Gamma chunk metadata
- Primary chromaticities chunk metadata
- sRGB chunk metadata
- ICC profile chunk metadata
- Background chunk metadata
- Significant bits chunk metadata

- Suggested palette chunk metadata
- Palette histogram chunk metadata
- Modified time chunk metadata
- Text chunk metadata
- Compressed textual data chunk metadata
- International textual data chunk metadata
- Calibration of pixel values chunk metadata
- Physical scale chunk metadata
- GIF application extension chunk metadata
- GIF Graphic Control Extension chunk metadata
- Image offset chunk metadata
- The rest chunk metadata

PNG header chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“IHDR”	0x49484452	LEVEL_START	0	NULL	0	TRUE
“Width”	0x49484452	VALUE_ITEM	UINT32		1	TRUE
“Height”	0x49484452	VALUE_ITEM	UINT32		1	TRUE
“BitDepth”	0x49484452	VALUE_ITEM	UINT8		1	TRUE
“ColorType”	0x49484452	VALUE_ITEM	UINT8		1	TRUE
“CompressionType”	0x49484452	VALUE_ITEM	UINT8		1	TRUE
“FilterType”	0x49484452	VALUE_ITEM	UINT8		1	TRUE
“InterlaceType”	0x49484452	VALUE_ITEM	UINT8		1	TRUE
“IHDR”	0x49484452	LEVEL_END	0	NULL	0	TRUE

Physical dimension chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“pHYs”	0x70485973	LEVEL_START	0	NULL	0	TRUE
“XAxis”	0x70485973	VALUE_ITEM	UINT32		1	FALSE
“YAxis”	0x70485973	VALUE_ITEM	UINT32		1	FALSE
“UnitSpecifier”	0x70485973	VALUE_ITEM	UINT8		1	FALSE
“pHYs”	0x70485973	LEVEL_END	0	NULL	0	TRUE

Transparency chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“tRNS”	0x74524e53	VALUE_ITEM	UINT8 or UINT16		Variable	FALSE

Gamma chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“gAMA”	0x67414d41	VALUE_ITEM	UINT32		1	FALSE

Primary chromaticities chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“cHRM”	0x6348524d	LEVEL_START	0	NULL	0	TRUE
“WhitePoint”	0x6348524d	LEVEL_START	0	NULL	0	TRUE

“x”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“y”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“WhitePoint”	0x6348524d	LEVEL_END	0	NULL	0	TRUE
“Red”	0x6348524d	LEVEL_START	0	NULL	0	TRUE
“x”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“y”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“Red”	0x6348524d	LEVEL_END	0	NULL	0	TRUE
“Green”	0x6348524d	LEVEL_START	0	NULL	0	TRUE
“x”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“y”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“Green”	0x6348524d	LEVEL_END	0	NULL	0	TRUE
“Blue”	0x6348524d	LEVEL_START	0	NULL	0	TRUE
“x”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“y”	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
“Blue”	0x6348524d	LEVEL_END	0	NULL	0	TRUE
“cHRM”	0x6348524d	LEVEL_END	0	NULL	0	TRUE

sRGB chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“sRGB”	0x73524742	VALUE_ITEM	UINT32		1	FALSE

ICC profile chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“iCCP”	0x70485973	LEVEL_START	0	NULL	0	TRUE
“ProfileName”	0x70485973	VALUE_ITEM	STRING		Variable	FALSE
“ProfileData”	0x70485973	VALUE_ITEM	RAW_DATA		Variable	FALSE
“iCCP”	0x70485973	LEVEL_END	0	NULL	0	TRUE

Background chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“bKGD”	0x624b4744	VALUE_ITEM	UINT8 or UINT16		Variable	FALSE

Significant bits chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“sBIT”	0x73424954	VALUE_ITEM	UINT8		Variable	FALSE

Suggested palette chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“sPLT”	0x73504c54	LEVEL_START	0	NULL	0	TRUE
“PaletteName”	0x73504c54	VALUE_ITEM	STRING		Variable	TRUE
“SampleDepth”	0x73504c54	VALUE_ITEM	UINT8		1	FALSE
“PaletteEntryCount”	0x73504c54	VALUE_ITEM	UINT16	<N>	1	FALSE

Palette entry level 0						
...						
Palette entry level n						
“sPLT”	0x70485973	LEVEL_END	0	NULL	1	FALSE

Palette entry level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“PaletteEntry”	<Entry No.>	LEVEL_START	0	NULL	0	TRUE
“Red”	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
“Green”	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
“Blue”	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
“Alpha”	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
“Frequency”	<Entry No.>	VALUE_ITEM	UINT16		1	FALSE
“PaletteEntry”	<Entry No.>	LEVEL_END	0	NULL	0	TRUE

Palette histogram chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“hIST”	0x68495354	VALUE_ITEM	UINT16		Variable	FALSE

Modified time chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“tIME”	0x74494d45	LEVEL_START	0	NULL	0	TRUE
“Year”	0x74494d45	VALUE_ITEM	UINT16		1	FALSE
“Month”	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
“Day”	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
“Hour”	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
“Minute”	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
“Second”	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
“tIME”	0x74494d45	LEVEL_END	0	NULL	0	TRUE

Text chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“tEXt”	0x74455874	LEVEL_START	0	NULL	0	TRUE
“Keyword”	0x74455874	VALUE_ITEM	STRING		Variable	FALSE
“Text”	0x74455874	VALUE_ITEM	STRING		Variable	FALSE
“tEXt”	0x74455874	LEVEL_END	0	NULL	0	TRUE

Compressed textual data chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“zTXt”	0x7a545874	LEVEL_START	0	NULL	0	TRUE
“Keyword”	0x7a545874	VALUE_ITEM	STRING		Variable	FALSE
“CompressionMethod”	0x7a545874	VALUE_ITEM	UINT8		1	FALSE
“Text”	0x7a545874	VALUE_ITEM	STRING		Variable	FALSE

“zTXt”	0x7a545874	LEVEL_END	0	NULL	0	TRUE

International textual data chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“zTXt”	0x69545874	LEVEL_START	0	NULL	0	TRUE
“Keyword”	0x69545874	VALUE_ITEM	STRING		Variable	FALSE
“CompressionFlag”	0x69545874	VALUE_ITEM	BOOL			
“Compression Method”	0x69545874	VALUE_ITEM	UINT8		1	FALSE
“LanguageTag”	0x69545874	VALUE_ITEM	STRING			
“TranslatedKeyword”	0x69545874	VALUE_ITEM	RAW_DATA			
“Text”	0x69545874	VALUE_ITEM	RAW_DATA		Variable	FALSE
“zTXt”	0x69545874	LEVEL_END	0	NULL	0	TRUE

Calibration of pixel values chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“pCAL”	0x7043414c	LEVEL_START	0	NULL	0	TRUE
“CalibrationName”	0x7043414c	VALUE_ITEM	STRING		Variable	FALSE
“OriginalZero(x0)”	0x7043414c	VALUE_ITEM	INT32		1	FALSE
“OriginalMax(x1)”	0x7043414c	VALUE_ITEM	INT32		1	FALSE
“EquationType”	0x7043414c	VALUE_ITEM	UINT8		1	FALSE
“NumberOfParameters”	0x7043414c	VALUE_ITEM	UINT8	<N>	Variable	FALSE

“UnitName”	0x7043414c	VALUE_ITEM	STRING		Variable	FALSE
“CalibrationParameter”	0	VALUE_ITEM	STRING		Variable	FALSE
...						
“CalibrationParameter”	<N> - 1	VALUE_ITEM	STRING		Variable	FALSE
“pCAL”	0x7043414c	LEVEL_END	0	NULL	0	TRUE

Physical scale chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“sCAL”	0x7343414c	LEVEL_START	0	NULL	0	TRUE
“UnitSpecifier”	0x7343414c	VALUE_ITEM	UINT8		1	FALSE
“PixelWidth”	0x7343414c	VALUE_ITEM	STRING		Variable	FALSE
“PixelHeigt”	0x7343414c	VALUE_ITEM	STRING		Variable	FALSE
“sCAL”	0x7343414c	LEVEL_END	0	NULL	0	TRUE

GIF application extension chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“gIFx”	0x67494678	LEVEL_START	0	NULL	0	TRUE
“ApplicationIdentifier”	0x67494678	VALUE_ITEM	STRING		Variable	FALSE
“AuthenticationCode”	0x67494678	VALUE_ITEM	UINT8		1	FALSE
“ApplicationData”	0x67494678	VALUE_ITEM	RAW_DATA		Variable	FALSE
“gIFx”	0x67494678	LEVEL_END	0	NULL	0	TRUE

GIF Graphic Control Extension chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“gIFg”	0x67494674	LEVEL_START	0	NULL	0	TRUE
“DisposalMethod”	0x67494674	VALUE_ITEM	UINT8		1	FALSE
“UserInputFlag”	0x67494674	VALUE_ITEM	UINT8		1	FALSE
“DelayTime”	0x67494674	VALUE_ITEM	UINT16		1	FALSE
“gIFg”	0x67494674	LEVEL_END	0	NULL	0	TRUE

Image offset chunk metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
“oFFs”	0x6f464673	LEVEL_START	0	NULL	0	TRUE
“XPosition”	0x6f464673	VALUE_ITEM	UINT32		1	FALSE
“YPosition”	0x6f464673	VALUE_ITEM	UINT32		1	FALSE
“UnitSpecifier”	0x6f464673	VALUE_ITEM	UINT8		1	FALSE
“oFFs”	0x6f464673	LEVEL_END	0	NULL	0	TRUE

The rest chunk metadata

The rest chunk data are passed as raw data

Name	Id	Type	Value Type	Value	Value Length	Read Only
<Chunk Name>	<Chunk ID>	VALUE_ITEM	RAW_DATA		Variable	FALSE

<Chunk Name> is a string representation of the chunk type value.

<Chunk ID> is a binary representation of the chunk type value.

Callback required for writing PNG metadata items

Value of these metadata items can be changed using **LPAFT_IG_METAD_ITEM_SET_CB** callback only:

Item Name	Item Id
Physical dimension chunk metadata	
“XAxis”	0x70485973
“YAxis”	0x70485973
“UnitSpecifier”	0x70485973

The rest PNG metadata items can be written using **LPAFT_IG_METAD_ITEM_ADD_CB** callback only.

PNG Metadata Item Name and ID constants

NOTE: Please see file `metadata.h` for the full list of PNG Metadata **Item Name** and **Item Id** constants.

GIF Non-image Data Structure

GIF file format is complex and has different non-image data that can be stored before and after an image data. The GIF metadata design allows to read/write any GIF non-image data and prevents from misunderstanding with GIF extensions storing order.

The following metadata are always saved before an image data:

- GIF Logical Screen Descriptor level
- GIF Image Descriptor level
- GIF Global Color Table level
- GIF Local Color Table level

The metadata of GIF extensions that are inside GIF Extensions After Image level (between items with Name **“AfterImageExtensions”** and Id `0xFFF` (Type `LEVEL_START` and `LEVEL_END`)) are saved after an image data.

All other GIF Extensions Metadata are saved before an image data.

NOTE: You can work with GIF metadata only when the ImageGear LZW Component is attached to the core ImageGear module.

GIF metadata level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“GIF”	IG_FORMAT_GIF	LEVEL_START	0	NULL	0	TRUE
GIF header level						
GIF Logical Screen Descriptor level						
GIF Image Descriptor level						
optional Global and/or Local Color Tables and optional GIF Extensions levels (extensions before image)						
“GIF”	IG_FORMAT_GIF	LEVEL_END	0	NULL	0	TRUE

All items between items with Name “GIF” and Id IG_FORMAT_GIF (Type LEVEL_START and LEVEL_END) are interpreted as GIF data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

GIF header level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“GIFHeader”	0x10	LEVEL_START	0	NULL	0	TRUE
“Signature”	0x101	VALUE_ITEM	STRING	“GIF”	3	TRUE
“Version”	0x102	VALUE_ITEM	STRING		3	TRUE
“GIFHeader”	0x10	LEVEL_END	0	NULL	0	TRUE

GIF Logical Screen Descriptor level

Name	Id	Type	Value Type	Value	Value Length	Read Only

“LogicalScreen Descriptor”	0x20	LEVEL_START	0	NULL	0	TRUE
“LogicalScreen Width”	0x201	VALUE_ITEM	UINT16		1	FALSE
“LogicalScreen Height”	0x202	VALUE_ITEM	UINT16		1	FALSE
“Fields”	0x205	LEVEL_START	0	NULL	0	TRUE
“GlobalColorTa bleFlag”	0x206	VALUE_ITEM	BOOL		1	FALSE
“ColorResoluti on”	0x207	VALUE_ITEM	UINT8		1	FALSE
“SortFlag”	0x208	VALUE_ITEM	BOOL		1	FALSE
“GlobalColorTa bleSize”	0x209	VALUE_ITEM	UINT8		1	FALSE
“Fields”	0x205	LEVEL_END	0	NULL	0	TRUE
“BackgroundC olorIndex”	0x203	VALUE_ITEM	UINT8			FALSE
“PixelAspectRa tio”	0x204	VALUE_ITEM	UINT8			FALSE
“LogicalScreen Descriptor”	0x20	LEVEL_END	0	NULL	0	TRUE

GIF Global Color Table level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“GlobalColorTa ble”	0x30	VALUE_ITEM	RAW_D ATA		Variable	TRUE

GIF Image Descriptor level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“ImageDescriptor”	0x2C	LEVEL_START	0	NULL	0	TRUE
“ImageLeftPosition”	0x2C1	VALUE_ITEM	UINT16		1	FALSE
“ImageTopPosition”	0x2C2	VALUE_ITEM	UINT16		1	FALSE
“ImageWidth”	0x2C3	VALUE_ITEM			1	FALSE
“ImageHeight”	0x2C4	VALUE_ITEM			1	FALSE
“Fields”	0x2C5	LEVEL_START	0	NULL	0	TRUE
“LocalColorTableFlag”	0x2C6	VALUE_ITEM	BOOL		1	FALSE
“InterlaceFlag”	0x2C7	VALUE_ITEM	BOOL		1	FALSE
“SortFlag”	0x2C8	VALUE_ITEM	BOOL		1	FALSE
“LocalColorTableSize”	0x2C9	VALUE_ITEM	UINT8		1	FALSE
“Fields”	0x2C5	LEVEL_END	0	NULL	0	TRUE
“ImageDescriptor”	0x2C	LEVEL_END	0	NULL	0	TRUE

GIF Local Color Table level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“LocalColorTable”	0x40	VALUE_ITEM	RAW_DATA		Variable	TRUE

GIF Extensions Metadata

GIF Graphic Control Extension level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“GraphicControlExtension”	0xF9	LEVEL_START	0	NULL	0	TRUE
“Fields”	0xF91	LEVEL_START	0	NULL	0	TRUE
“DisposalMethod”	0xF92	VALUE_ITEM	UINT8		1	FALSE
“UserInputFlag”	0xF93	VALUE_ITEM	BOOL		1	FALSE
“TransparentColorFlag”	0xF94	VALUE_ITEM	BOOL		1	FALSE
“Fields”	0xF91	LEVEL_END	0	NULL	0	TRUE
“DelayTime”	0xF95	VALUE_ITEM	UINT16		1	FALSE
“TransparentColorIndex”	0xF96	VALUE_ITEM	UINT8		1	FALSE
“GraphicControlExtension”	0xF9	LEVEL_END	0	NULL	0	TRUE

GIF Comment Extension level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“CommentExtension”	0xFE	VALUE_ITEM	STRING		Variable	TRUE

GIF Application Extension level

Name	Id	Type	Value Type	Value	Value Length	Read Only

“ApplicationExtension”	0xFF	LEVEL_START	0	NULL	0	TRUE
“ApplicationIdentifier”	0xFF1	VALUE_ITEM	STRING		8	FALSE
“Appl.AuthenticationCode”	0xFF2	VALUE_ITEM	UINT8		4	FALSE
“ApplicationData”	0xFF3	VALUE_ITEM	RAW_DATA		Variable	FALSE
“ApplicationExtension”	0xFF	LEVEL_END	0	NULL	0	TRUE

GIF Plain Text Extension level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“PlainTextExtension”	0x01	LEVEL_START	0	NULL	0	TRUE
“TextGridLeftPosition”	0x11	VALUE_ITEM	UINT16		1	FALSE
“TextGridTopPosition”	0x12	VALUE_ITEM	UINT16		1	FALSE
“TextGridWidth”	0x13	VALUE_ITEM	UINT16		1	FALSE
“TextGridHeight”	0x14	VALUE_ITEM	UINT16		1	FALSE
“CharacterCellWidth”	0x15	VALUE_ITEM	UINT8		1	FALSE
“CharacterCellHeight”	0x16	VALUE_ITEM	UINT8		1	FALSE
“TextForegroundColorIndex”	0x17	VALUE_ITEM	UINT8		1	FALSE
“TextBackgroundColorIndex”	0x18	VALUE_ITEM	UINT8		1	FALSE
“PlainTextData”	0x19	VALUE_ITEM	STRING		Variable	FALSE

GIF Extensions After Image level

Name	Id	Type	Value Type	Value	Value Length	Read Only
“AfterImageExtensions”	0xFFF	LEVEL_START	0	NULL	0	TRUE
One or more GIF extension levels						
“AfterImageExtensions”	0xFFF	LEVEL_END	0	NULL	0	TRUE

Callback required for writing GIF metadata items

The GIF Logical Screen Descriptor level and GIF Image Descriptor level metadata items can be written using **`LPAFT_IG_METAD_ITEM_SET_CB`** callback function.

Other GIF metadata can be written using **`LPAFT_IG_METAD_ITEM_ADD_CB`** callback function.

GIF Metadata Item Name and ID constants

NOTE: Please see file `metadata.h` for the full list of GIF Metadata **Item Name** and **Item Id** constants.

DCRAW Non-Image Data Structure

- Common level
- Format-specific level
- Statistics

DCRAW level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"Common"	IGMDTAG_ID_DCRAW_COMMON	LEVEL_START	0	NULL	0	TRUE

“EXIF”	IG_FORMAT_EXIF_TIFF	LEVEL_START	0	NULL	0	TRUE
“TIFF”	IG_FORMAT_TIF	LEVEL_START	0	NULL	0	TRUE
"CIFF"	IG_FORMAT_CIFF	LEVEL_START	0	NULL	0	TRUE
“Statistics”	IGMDTAG_ID_DCRAW_STATISTICS	LEVEL_START	0	NULL	0	TRUE

Common level

The Common sub-tree contains values that are common or useful for all raw files.

All values in the Common sub-tree have Type = VALUE_ITEM and are Read Only.

Name	Id	Description
ImageWidth	IGMDTAG_ID_DCRAW_IMAGEWIDTH	Image width after that masked pixels were trimmed out
ImageHeight	IGMDTAG_ID_DCRAW_IMAGEHEIGHT	Image height after that masked pixels were trimmed out
BitsPerSample	IGMDTAG_ID_DCRAW_BITSPERSAMPLE	Bits per sample in the original raw image (before reconstruction)
PhotometricInterpretation	IGMDTAG_ID_DCRAW_PHOTOMETRICINTERPRETATION	TIF_PHOTO_CFA, TIF_PHOTO_RGB
SamplesPerPixel	IGMDTAG_ID_DCRAW_SAMPLESPERPIXEL	Number of samples per pixel in the original image, before reconstruction
Make	IGMDTAG_ID_DCRAW_MAKE	Name of the camera manufacturer
Model	IGMDTAG_ID_DCRAW_MODEL	Name of the camera model
UniqueCameraModel	IGMDTAG_ID_DCRAW_UNIQUECAMERAMODEL	Unique, non-localized name for the camera model that created the image in the raw file.
TimeStamp	IGMDTAG_ID_DCRAW_TIMESTAMP	Time stamp

CFARepeatPatternDim	IGMDTAG_ID_DCRAW_CFAREPEATPATTERNDIM	This tag encodes the number of pixels horizontally and vertically that are needed to uniquely define the repeat pattern of the color filter array.
CFAPattern	IGMDTAG_ID_DCRAW_CFAPATTERN	This tag encodes the actual color filter array geometric pattern of the image sensor used to capture the single sensor color image.
BlackLevelRepeatDim	IGMDTAG_ID_DCRAW_BLACKLEVELREPEATDIM	This tag specifies repeat pattern size for the “BlackLevel” tag.
AsShotNeutral	IGMDTAG_ID_DCRAW_ASSHOTNEUTRAL	Specifies the selected white balance at time of capture, encoded as the coordinates of a perfectly neutral color in linear reference space values.
WhiteLevel	IGMDTAG_ID_DCRAW_WHITELEVEL	Image pixel value that corresponds to fully saturated encoding level.
CalibrationIlluminant1	IGMDTAG_ID_DCRAW_CALIBRATIONILLUMINANT1	The illuminant used for ColorMatrix1 color conversion matrix. Allowed values are the same as allowed values for LightSource EXIF tag (see Exif specification).
CalibrationIlluminant2	IGMDTAG_ID_DCRAW_CALIBRATIONILLUMINANT1	The illuminant used for ColorMatrix2 color conversion matrix. Allowed values are the same as allowed values for LightSource EXIF tag (see Exif specification).
ColorMatrix1	IGMDTAG_ID_DCRAW_COLORMATRIX1	Conversion matrix from XYZ to camera colorspace, for illuminant specified by CalibrationIlluminant1 tag. Matrix is stored in row scan order.
ColorMatrix2	IGMDTAG_ID_DCRAW_COLORMATRIX2	Conversion matrix from XYZ to camera colorspace, for illuminant specified by CalibrationIlluminant2 tag. Matrix is stored in row scan order.
BaselineExposure	IGMDTAG_ID_DCRAW_BASELINEEXPOSURE	Specifies baseline exposure of the camera, in EV units. Application may use this value to adjust exposure compensation for particular camera model.

Format-specific level

This level includes format-specific metadata that is found in the image (EXIF, TIFF or CIFF). See corresponding sections for structure of metadata for these formats.

Statistics

This section contains data which is collected during image loading, rather than obtained from image's header or metadata. Only image loading functions, such as IG_fltr_load_file generate this section. Header reading functions, such as IG_info_get, do not generate it.

For some cameras ImageGear uses predefined values for BlackLevel, so they are not calculated from pixel data. In this case, ImageGear also places BlackLevel tag to Statistics section, rather than in Common section, for uniformity reasons.

The following tags can be present in the Statistics section:

Name	ID	Description
BlackLevelRepeatDim	IGMDTAG_ID_DCRAW_BLACKLEVELREPEATDIM	This tag specifies repeat pattern size for the "BlackLevel" tag.
BlackLevel	IGMDTAG_ID_DCRAW_BLACKLEVEL	This tag specifies the zero light (a.k.a. thermal black or black current) encoding level. It is calculated during image loading by averaging pixel values in the masked areas of the image.

HD Photo format uses the same non-image data storage format as is used by TIFF. However, HD Photo uses its own set of tags for image-related parameters, such as image width, height and pixel format. It also uses a few descriptive TIFF tags such as Copyright and Artist. HD Photo non-image data may include EXIF, EXIF GPS and IPTC subtrees.

Glossary

This glossary contains a mixture of terminology used by AccuSoft in both its software products and its documentation. Since these terms come from many different disciplines and because many of the terms have different meanings in each discipline, each glossary entry is followed by the name of the field from which the definition is taken.

For a detailed description, refer to the references listed in the Bibliography.

To find terms which start with a numeral (0-9), look under its spelling, for example, the term “8-bit gray level” can be found as - “eight bit gray level.”

absolute coordinates (imaging)

Absolute coordinates refer to a common origin, for example, the upper left corner of a display screen. This is the opposite of relative coordinates.

ACCUAPI (AccuSoft)

AccuSoft **A**pplication **P**rogram **I**nterface. See “API (software)” on page -559.

additive primary colors (imaging)

Red, Green, Blue - the 3 colors used to create all other colors when direct, or transmitted light is used (as in a video monitor). They are called additive primaries, because when these three colors are superimposed they produce white.

anti-aliasing (imaging)

A method of filling in data that is missing due to under-sampling. In imaging, this usually involves the process of removing jagged edges by interpolating values in-between pixels of contrast. These methods are most often used to remove or reduce the stair-stepping artifact found in digital high contrast images.

AOI (Image Processing)

Area **O**f **I**nterest. An area of interest is a rectangle within an image defined as two points within the image. An AOI can be written as (x1,y1)-(x2,y2). All AOIs are parallel with the image’s axes. See “ROI (AccuSoft image processing)” on page -575.

API (software)

Application **P**rogrammer’s **I**nterface. The set of routines that make up a library or toolkit. Some times called a binding.

aspect ratio (imaging)

The proportion of an image’s size given in terms of the horizontal length verses the vertical height. An aspect ratio of 4:3 indicates that the image is 4/3 times as wide as it is high.

Bezier curve (graphics)

A curve created from endpoints and two or more control points that serve as positions for the shape of the curve. Originated by P. Bezier (~1962) for the use in car body descriptions.

bit block transfer

A raster operation that moves a block of bits representing a portion of an image or scene from one location in the frame buffer to another. Usually written as “bit blt”.

bin (image processing)

See “histogram (imaging)” on page -568.

Bit Block Transfer (Windows)

An optimized movement of a large block of computer memory from one location to another. Used for moving images or sub-images to and from areas of computer memory.

bit blt

bit_block_transfer.

bitmap (imaging)

An image is a bitmap if it contains a value for each of its pixels. This is the opposite of vector images where a small set of values generate an object.

bit plane (imaging)

A hypothetical 2-D plane containing a single bit of memory for each pixel in a image. If each 8-bit pixel is thought of as a stack of 8 coins, and an image as many rows and columns of these stacked coins then the 3rd bit plane would be the plane consisting of the 3rd coin from each stack.

bounding rectangle (geometry)

The smallest rectangle that fits around a given object. In imaging, the rectangle is usually rotationally restricted to be parallel to both image axes.

.BMP (file format extension)

Format originator:Microsoft Corporation

16011 NE 36th Way, Box 97917

Redmond, WA 98073

Call-back function (software)

A function that is passed to another function as a parameter. The function receiving the call-back function can call this function. This is used to change the behavior of a given routine without knowing beforehand what it is expected to do.

Cartesian coordinates (imaging)

A 2-dimensional equally spaced grid iron that uniquely assigns every point in the plane, (one and only one), co-ordinate pair; (x, y). In imaging, each point is usually referred to as a pixel and the x and y values take on integer values. Most images use the top-left as the (0,0), or origin. See “coordinates” on page -563.

Chroma-key (image processing)

An image blending function that replaces pixels of a specified hue range with pixels from a second image. This is often referred to the weatherman effect because most weather forecasters use a solid blue or green background to make it look as if they are standing in front of a huge weather map. It is important to remember that it is the hue that is used in the blending function and not the intensity or saturation.

C.I.E (color imaging)

Commission **I**nternationale de l’**E**clairage. (International Commission of Illumination). A standards organization which provides specifications for the description of device independent color.

clipboard (Windows)

The clipboard is a windows data structure used to exchanged data between applications. It is a common area where applications place data and others can access it. These operations are usually referred to as Cut (place data in) and Paste (take data out).

closing (image processing)

See “MPEG (image compression)” on page -572.

CMY & CMYK

Cyan, **M**agenta, **Y**ellow, **(K) black**. Computer monitors are additive, but color printers are subtractive. Instead of combining light from monitor phosphors, printers coat paper with colored pigment that removes specific colors from the illumination light.

CMY is the subtractive color model that corresponds to the additive RGB model. Cyan, magenta, and yellow are the color complements of red, green, and blue. Due to the difficulties of manufacturing pigments that produce black when mixed together, a separate black ink is often used and is referred to as K (‘B’ is already used for blue).

color map (imaging)

See “Look-Up-Table (computer hardware)” on page -571

color model (imaging)

See “color space (imaging)” on page -562.

color space (imaging)

A mathematical coordinate system (space) for assigning numerical values to colors. There are many ways to define such spaces, each with its own benefits and problems.

See Also:

- “CMY & CMYK” on page -561
- “HIS (color imaging)” on page -569
- “HLS (color imaging)” on page -569
- “HSV (color imaging)” on page -569
- “RGB (imaging)” on page -575
- “YIQ (color imaging)” on page -578

compression (imaging)

An image processing method for saving valuable disk and memory space by reducing the amount of space required to save a digital image. The graphics data is rewritten allowing it to be represented by a smaller set of data. Do not confuse this with encoding. See “lossless (image compression)” on page -571 and“lossy (image compression)” on page -571.

compression ratio (imaging)

The ratio of a file’s uncompressed size over its compressed size.

concave (geometry)

A 2-dimensional blob, for example, a **r**egion **o**f **i**nterest (ROI), where at least one tangent is drawn that touches the blob at two different locations, and there is a point on the tangent between the two contacts that does not touch the blob.

In simpler words, if a rubber band could be snugly wrapped around a concave blob there would be places where the rubber band lifts off and does not touch the blob. Concave is the opposite of convex.

convex (geometry)

A 2-dimensional blob, for example, a **r**egion **o**f **i**nterest (ROI), where every tangent that can be drawn touches the blob at a continuous stretch of the blob’s surface with no gaps.

In simpler words, if a rubber band could be snugly wrapped around a convex blob there would be no places where the rubber band lifts off and is not touching the blob. Convex is the opposite of concave.

convolution (image processing)

An image processing operation that is used to spatially filter an image. A convolution is defined by a kernel that is a small matrix of fixed numbers. The size of the kernel, the numbers within it, and a single normalizer value define the

operation that is applied to the image. The kernel is applied to the image by placing the kernel over the image to be convolved and sliding it around to center it over every pixel in the original image.

At each placement the numbers (pixel values) from the original image are multiplied by the kernel number that is currently aligned above it. The sum of all these products is tabulated and divided by the kernel's normalizer. This result is placed into the new image at the position of the kernel's center. The kernel is translated to the next pixel position and the process repeats until all image pixels have been processed.

As an example, a 3x3 kernel holding all '1's with a normalizer of 1/9 performs a neighborhood averaging operation. Each pixel in the new image is the average of its 9 neighbors from the original.

coordinates

A pair of numbers that represent a specific location in a two-dimensional plane, for example, an image or on a map.

See Also:

- "absolute coordinates (imaging)" on page -559
- "device coordinates (imaging)" on page -564
- "Cartesian coordinates (imaging)" on page -561
- "polar coordinates (imaging)" on page -574
- "relative coordinates" on page -575
- "screen coordinates (imaging)" on page -576
- "world coordinates" on page -577

crop (Imaging)

An image processing method for removing the region near the edge of the image, but keeping the central area.

.DCX (file format extension)

Format originator: Intel

DDB (Windows)

Device-Dependent Bitmap. A Window image specification that depends on the capabilities of a specific graphics display controller. Since a DDB is matched to the current graphics controller, it is fast and easy to display since large blocks of memory need only be copied to the controller.

See Also:

"DIB (Windows)" on page -564.

decompression (imaging)

When an image or other digital data set is compressed and stored, it is not usable until it is decompressed into it original form.

device coordinates (imaging)

The co-ordinates of the coordinate system that describe the physical units that defines the computer screen.

device dependent (software)

Software written to work on a specific set of hardware platforms. Since these routines make use of physical device attributes, they may behave differently on other devices, although they will most often not work on other devices.

See Also:

- “device independent (software)” on page -564
- “DIB (Windows)” on page -564.

device driver (software)

A set of low-level software routines that work with and control a specific hardware device. The names and functions are often standardized across many similar devices. This allows higher level software to use the hardware as a generic device. This frees the higher-level software from dealing with the particulars of specific devices and allows devices to be interchanged.

device independent (software)

Software or data structures that are designed to work with or on a wide set of hardware platforms.

See Also:

- “device independent (software)” on page -564
- “DIB (Windows)” on page -564.

DIB (Windows)

Device-Independent Bitmap is a Windows-defined image format specification. It is called device-independent because of its straightforward, common-denominator, format. It has all the information that a basic digital image needs and is laid out in a simple specification. Its simplicity makes it an ideal format for holding images that need to be shared by several programs.

See Also:

- “DDB (Windows)” on page -563
- the book *Programming Windows* by Charles Petzold.

dilation (image processing)

See “MPEG (image compression)” on page -572.

dithering (imaging)

The method of using neighborhoods of display pixels to represent one image intensity or color. This method allows low-intensity resolution display devices to simulate higher resolution images. For example, a binary laser printer can use block patterns to display gray-scale images.

See Also:

"halftone (imaging)" on page -568.

DLL (Microsoft Windows)

Dynamic **L**inked **L**ibrary. A compiled and linked collection of computer functions that are not directly bound to an executable. These libraries are linked at run-time by Windows. Since Windows is in charge of managing (loading, linking, and removing) the DLLs, they are available to all executables currently running. Each executable links to a shared DLL saving memory by avoiding redundant functions from co-existing. DLLs allow a new level of modularity by providing a means to modify and update executables without re-linking. Just copy a new version of the DLL to the correct disk directory.

DPI (printing)

Dots **P**er **I**nch. The number of printer dots that can be printed in one inch. The printer's resolution is defined by the number of dots per inch: lower resolution = less dots per inch, higher resolution = more dots per inch.

edge (image processing)

In an image, an edge is a region of contrast or color change. Edges are useful in machine vision since optical edges often mark the boundary of physical objects.

edge detection (image processing)

A method that isolates and locates an optical edge in a digital image.

edge map (image processing)

An edge map is the output of an image-processing filter that transforms an image into an image where intensity represents a change in the contrast (optical edge) of the original image.

(eight) 8-bit image (digital imaging)

An image where each pixel has 8-bits of information. An 8-bit pixel contains one of 256 possible values. There are two common types of 8-bit images: gray scale and indexed color.

In a gray-scale image, each pixel takes one of 256 shades of gray and the shades are linearly distributed from 0 (black) to 256 (white). An 8-bit gray-scale image does not require a palette but may have one.

An indexed color image is always a palette image. Each pixel is used as an index to the palette. These images can have up to 256 different colors. This includes hues as well as shades. Indexed 8-bit images are good for low color resolution images that do not need processing. They are 3 times smaller than full-color RGB

images, but because the pixel values are not linear, many image-processing algorithms cannot work with them. They must be promoted to 24-bit for image processing.

8-bit gray level (AccuSoft term)

This indicates 8-bit gray scale. 8-bit gray level is used to distinguish between 8-bit indexed color (8i) and 8 bit gray-scale. An 8-bit gray level DIB image is one where each pixel in the bitmap is unchanged by its palette when displayed. Each palette entry is the same as its index.

8i (AccuSoft term)

This indicates 8-bit indexed color. 8i is used throughout this manual to distinguish between 8-bit gray scale (8-bit gray level) and 8-bit indexed color. An 8-bit indexed color DIB is one where each 8-bit pixel value in the bitmap is used as an index to the palette.

The palette dictates which RGB color the pixel displays. These images are compact ways of storing color images. However they are difficult to process because the bytes that make up the pixel can no longer be ordered with any certainty.

Encoding

The format for storing uncompressed data (binary, ASCII, etc.), how it is packed (e.g. 4-bit pixels may be packed at a rate of two pixels per byte), and the unique set of symbols used to represent the range of data items.

.EPS (file format extension)

Format originator:Adobe Systems, Inc.

1585 Charleston Road

Mountain View, CA 94039

equalize (image processing)

An image-processing algorithm that redistributes the frequency of image pixel values allowing equal representation for any given continuous range of values. In an ideal world, an equalized image has the same number of pixels in the range from 10-20 as it does from 200-210. However, since digital images have quantized intensity values, the range totals are rarely identical but usually close.

erosion (image processing)

See “MPEG (image compression)” on page -572

file format (software)

A specification for holding computer data in a disk file. The format dictates what information is present in the file and how it is organized.

filter (image processing)

An image-processing filter is a transform that removes a specified quantity from an image. For instance a spatial filter removes high, medium or low spatial frequencies from an image.

(four) 4 bit image (digital imaging)

An image file format that allows 4-bits per pixel. This image can contain up to 16 (2^4) different colors or levels of gray.

frame (imaging)

A single picture, usually taken from a collection of images for example, a movie or video stream.

frame buffer (imaging hardware)

A computer peripheral that stores and sometimes manipulates digital images.

frame processes (image processing)

Image-processing algorithms that operate on a single image.

fx (imaging)

See “special effects (image processing)” on page -576.

gain & level (imaging)

Gain and level are image-processing terms that correspond to the brightness and contrast control on a television. The gain is the “contrast”, and the level is the “brightness.” By changing the level, the entire range of pixel values are linearly shifted brighter or darker. Gain on the other hand linearly stretches or shrinks the intensity range, altering the contrast.

gamma correction (imaging)

A non-linear function that is used to correct the inherent non-linearities of cameras and monitors. The intensity of the luminescent phosphor on the raster display is non-linear. Gamma correction is an adjustment to the pixel intensity values that make up for this inherent non-linearity.

geometric transform (image processing)

A class of image processing transforms that alter the location of pixels. This class includes rotates and warps.

.GIF (file format extension)

Name:Graphics Interchange File Format

Format originator:CompuServe Inc.

500 Arlington Center Blvd.

Columbus, OH 43220

This format uses the LZW compression created by Unisys. It is the same as the LZW compression used in the TIFF file format, except that the bytes are reversed and the string table is upside-down.

All GIF files have a palette. Some GIF files can be interlaced - the raster lines can appear as every 4 lines, then every 8 lines, then every other line. This is due to GIF files usually being received from a modem.

GUI

Graphical **U**ser **I**nterface. A computer-user interface that uses graphical objects and a mouse for user interaction, for example Microsoft Windows.

graphics library (software)

A collection of software routines that work on digital images. These collections usually contain routines for drawing various graphical objects, for example, lines, circles, and rectangles.

gray level (imaging)

A shade of gray assigned to a pixel. The shades are usually positive integer values taken from the gray-scale. In an 8-bit image a gray level can have a value from 0 to 255.

gray scale (imaging)

A range of gray levels. Zero is usually black and higher numbers indicate brighter pixels.

group III Fax (Imaging compression)

A CCITT standard for transmission of facsimile data. It compresses black and white images using a combination of differential, run length and Huffman coding.

halftone (imaging)

The reproduction of a continuous-tone image on a device that does not directly support continuous output. This is done by displaying or printing a pattern of small dots that simulate the desired output color or intensity. These methods are used extensively in magazines and newspapers.

handle (software)

A handle references a data object. A handle is a type of pointer but it usually contains, internally, more information about the referenced object.

histogram (imaging)

A tabulation of pixel value populations displayed as a bar chart where the x-axis represents all the possible pixel values and the y-axis is the total image count of each given pixel value. A histogram counts how many pixels in the image have a given intensity value or range of values.

Each histogram intensity value or range of values is called a bin. Each bin contains a positive number that represents the number of pixels in the image that fall within the bin’s range. A typical 8-bit gray-scale histogram contains 256 bins. Each bin has a range of a single intensity value. Bin 0 contains the number of

pixels in the image that have a gray-scale value of 0 or black; bin 255 contains the number of white (255) pixels. When the collection of bins are sorted (0-255) and charted, the graph displays the intensity distributions of all the images pixels.

HLS (color imaging)

Hue **S**aturation, and **L**ightness. A method that describes any color as a *triplet* of real values. The hue represents the color or wavelength of the color. It is sometimes called tone and is commonly known as color. The hue is taken from the standard color wheel and is calibrated in degrees.

Saturation is the depth of the color. It states how gray the color is. It is a real valued parameter from 0.0 to 1.0 with 0.0 indicating full gray and 1.0 representing pure hue.

Lightness determines how black or white a color is. It ranges from 0.0 to 1.0 but with 0.0 representing black and 1.0 white. A lightness of 0.5 is a pure hue.

HSV (color imaging)

Hue, **S**aturation, and **V**alue.

Huffman coding (image compression)

A method of encoding symbols that varies the length of the code in proportion to its information content. Groups of pixels that appear frequently in a image are coded with fewer bits than those of lower occurrence.

HIS (color imaging)

Intensity, **H**ue, and **S**aturation.

image format (image storage)

There are many digital image formats. Some of these are: TIFF, DIB, GIF, and JPEG. The image format specification dictates which image information is present and how it is organized in memory. Many formats support various sub-formats or ‘flavors’.

image processing

The general term “image processing” refers to a computer discipline wherein digital images are the main data object. This type of processing can be broken down into several sub-categories: compression, image enhancement, image filtering, image distortion, image display and coloring, and image editing.

See Also:

“machine vision” on page -572.

indexed color image (imaging)

An image where each pixel value is used as an index to a palette for interpretation before the pixel is displayed. These images contain a palette that is initialized specifically for a given image. The pixel values are usually 8-bit and the palette 24-bit (8-red, 8-green, and 8-blue).

See Also:

“(eight) 8-bit image (digital imaging)” on page -565.

invert intensity (image processing)

An image processing operation where each pixel is subtracted from the maximum pixel value allowed. This produces a photographic negative of the original. For an 8-bit image the inverse function is:

invert(pix) = 255-pix;

For an 8-bit RGB image the function is:

invert(Rpix) = 255-Rpix;

invert(Gpix) = 255-Gpix;

invert(Bpix) = 255-Bpix;

“jaggies” (imaging)

A term used to describe the visual appearance of lines and shapes in raster pictures that results from a grid of insufficient spatial resolution.

JPEG JFIF (image compression)

Joint **P**hotographic **E**xperts **G**roup. A collaborative specification of the CCITT and the ISO for image compression. The standard JPEG compression algorithm, which is used by ImageGear, is a lossy compression scheme – it loses data.

.JPG (file format extension)

Format originator:Joint Photographics Experts Group

kernel (image processing)

A small matrix of pixels, usually no bigger that 9x9, that is used as an operator during image convolution. The kernel is set prior to the convolution in a fashion that emphasizes a particular feature of the image. Kernels are often used as spatial filters, each one tuned to a specific spatial frequency that the convolution is intended to highlight.

See Also:

convolution (image processing).

Lempel Ziff Welch (data compression)

A dictionary-based image compression method with lossless performance that results in fair compression ratios. Most files are compressed at 2:1.

level (imaging)

See gain & level (imaging).

library (software)

A collection of software functions that can be called upon by a higher level program. Most libraries are collections of similar routines, for example, those used for graphical or image processing.

See Also:

“DLL (Microsoft Windows)” on page -565.

Look-Up-Table (computer hardware)

A look-up-table or LUT is a continuous block of computer memory that computes the values of a function for one variable. The LUT is set up for the function's variable to be used as an address or offset into the memory block. The value that resides at this memory location becomes the function's output. Because the LUT values need only be initialized once, LUTs are very useful for image processing due to their inherent high speed.

LUT[pixel_value] = f(pixel_value)

LUTs come in various widths, usually in units of bits. An **nxm** bit LUT has 2^n addresses or 256 stored values. Each value is 2^m bits wide.

If the second dimension is left off it can be assumed to be equal to the first. In gray-scale image processing, LUTs are commonly 8x8, and the bit widths are usually assumed.

A linear LUT, sometimes called a NOP LUT or pass through, is a LUT that is initialized to output the same values as the input. NOP_LUT[pixel_value] = pixel_value.

See “palette (digital imaging)” on page -573.

lossless (image compression)

A method of image compression where there is no loss in quality when the image is uncompressed. The uncompressed image is mathematically identical to its original. Lossless compression is usually lower in compression ratio than lossy compression.

lossy (image compression)

A method of image compression where some image quality is sacrificed in exchange for higher compression ratios. The amount of quality degradation depends on the compression algorithm used and by a user-selected quality variable.

LUT (computer)

Look-**U**p-**T**able. See “Look-Up-Table (computer hardware)” on page -571.

LUT transform (image processing)

A LUT transform is an image processing method that takes an image and passes each pixel, one at a time, through a pre-set LUT. Each new pixel is a function of one and only one pixel from the original image and is arranged in the same location.

Any image-processing algorithm that transforms a single pixel into another single pixel, both from the same location, can be performed quickly using a LUT.

Square_root_LUT[pixel_value] = sqrt(pixel_value)

See Also:

"Look-Up-Table (computer hardware)" on page -571.

LZW (data compression)

Lempel **Z**iff **W**elch. See "Lempel Ziff Welch (data compression)" on page -570.

machine vision

A sub-discipline of artificial intelligence that uses video cameras or scanners to obtain information about a given environment. Machine vision processes extract information from digital images about objects in the image. This is the opposite of computer graphics that takes various data describing objects in and produces an output image. Machine vision takes an image in and outputs some level of description about the objects in it, (i.e. color, size, brightness).

See Also:

"image processing" on page -569.

matrix operation (image processing)

See "neighborhood process (image processing)" on page -573.

median filter (image processing)

An image spatial filtering operation based on an input pixel and its 8 neighbors. The resulting value is the median (5th from the sorted values). A median filter is often used to reduce spike or speckling noise from a gray-scale image. It has the advantage over convolution smoothing - it better preserves edges.

morphing (image processing)

An imaging process where one image is gradually transformed into a second image, where both images previously existed. The result is a sequence of in-between images when played sequentially, as in a film loop show, give the appearance of the starting image being transformed to the second image.

Morphing is made up of a collection of image processing algorithms. The two major groups are: warps and blends. Do not confuse this with morphology.

MPEG (image compression)

Motion **P**ictures **E**xperts **G**roup. An ISO specification for the compression of digital-broadcast quality full-motion video and sound.

neighborhood process (image processing)

A class of image-processing routines that works on neighborhoods of pixels. Each pixel in the new image is computed as a function of the neighborhood of the pixel from the original pixel. The neighborhood ID is defined by a kernel that is set once for each image to be processed.

See Also:

“point process (image processing)” on page -574.

(one) 1-bit image (digital imaging)

An image comprised of pixels that contain only a single bit of information. Each pixel is either on or off. Normally, “on” is white and “off” is black.

opening (image processing)

See “MPEG (image compression)” on page -572.

overlay (imaging)

An image or sub-image that can be placed over a given image. The pixels from the original image are not altered but the overlay can be viewed as if they had been. Usually used to place temporary text and annotation marks, for example, arrows on a image.

packed bits (imaging)

A binary image is usually stored in computer memory (8 pixels per byte). In this case each byte is referred to as being filled with packed bits. This saves space but makes reading and writing any individual pixel harder since most computers cannot directly access memory in chunks smaller than a byte.

palette (digital imaging)

A digital image palette is a collection of 3 look-up-tables, or LUTs, that are used to define a given pixel's display color. One LUT is for red, one for green and one for blue. The number of entries in the LUTs depend on the width (in bits) of the image’s pixels.

A palette image requires its palette in order to be displayed in a fashion that makes sense to the viewer. This is often the case for color 8-bit images. Without a palette describing what color each pixel needs for display, this type of image would most likely be displayed as randomly selected noise.

A gray-scale palette is one where each of the 3 LUTs are linear. The output is whatever is input to them. Since each color component (R, G, B) is an equal value, any pixels input to them are displayed in a varying shade of gray.

See Also:

“Look-Up-Table (computer hardware)” on page -571.

pattern recognition (imaging)

A sub-discipline of machine vision where images are searched for specific patterns. Optical character recognition or “OCR” is one type of pattern recognition, where images are searched for the letters of the alphabet.

.PCX (file format extension)

Format originator:ZSoft Corp.

450 Franklin Road Suite 100

Marietta, GA 30067

pixel (imaging)

An abbreviated version of the term **PI**cture (**X**) **EL**ement. This is the most fundamental element of a digital image. A digital image is made up of rows and columns of points of light. Each indivisible point of light is a pixel. Each pixel in an image is addressed by its column (x) and its row (y) usually written as the coordinate pair (x, y). An 8-bit pixel can take on one of 256 values. A 24-bit pixel has 3, 8-bit components for each of the primary colors, red, green, and blue.

point process (image processing)

A class of image processing transforms where every pixel is taken, one at a time from an image, and mathematically transformed into a new value with no input from any other pixel in the image. A point process is a degenerative neighborhood process where the kernel is a matrix of pixels that is 1x1 or in other words a single pixel.

polar coordinates (imaging)

An alternative to the usual Cartesian method of addressing image pixels. Polar coordinates use the coordinate pair, angle and radius from an origin instead of column and row.

posterize (imaging)

A special effect that decreases the number of colors or gray-scales in an image. The default image pixel contains 256 levels of gray or 256 levels of red, green, and blue. Using this effect reduces these numbers.

pseudocolor (image processing)

A method of assigning color to ranges of a gray-scale image’s pixel values. Most often used to highlight subtle contrast gradients or for visually quantifying pixel values. The applied color usually has no correspondence to the original scene. The colors are used only as a guide or highlight.

raster (imaging)

A term that describes a single row of a digital image. A raster image is made up of rows of pixels. This is opposed to vector images, where an image is made up of a list of polygon nodes. A raster is sometimes called a scan-line.

relative coordinates

Relative coordinates refer to position, as identified as the distance from a local origin.

render (imaging)

The process of displaying an image. The final and actual displayed image is said to be rendered.

resolution (imaging)

There are two types of resolution in digital images; spatial and intensity. Spatial resolution is the number of pixels per unit of length along the x and y axis. Intensity resolution is the number of quantized levels that a pixel can have.

RGB (imaging)

Red, **G**reen, **B**lue. A triplet of numeric values that describe a color.

RGBQUAD

Red, **G**reen, **B**lue, **Q**uad. A set of four numbers used to describe a color. The forth number is always set to zero. This creates an efficient color LUT or palette. It is more efficient because most computers find multiplying by 4 easier then by 3, as is the case in an RGB triplet.

ROI (AccuSoft image processing)

Region **O**f **I**nterest. A region of interest or ROI is a specification and date structure that allows for the definition of arbitrarily shaped regions within a given image, often called sub-images. A ROI can be thought of as a place holder which remembers a location within an image. ROIs are of several types, each defined in a manor that makes sense for its type.

ROIs are either a rectangle (also called an AOI), square, circle, or a segment list. A rectangle is defined by any two points in the image. From these two points one and only one rectangle can be drawn. A square is defined by a single point and a single length. A circle is defined by its center and radius. A segment list is an arbitrary list of triplets (x, y, xlen); a single point and a length to the right.

Every point in an image is either inside or outside of a given ROI.

Most image processing functions in this package work only within a given ROI. The ROI can encompass the entire image.

See Also:

“AOI (Image Processing)” on page -559.

scan line (imaging)

See “raster (imaging)” on page -574.

screen coordinates (imaging)

Screen coordinates are those of the actual graphics display controller. The origin is almost always at the upper left-hand corner of the display.

See Also:

“coordinates” on page -563.

segment (imaging)

A contiguous section of a raster line. It is defined in physical coordinates by the triplet of its left most point and length (x, y, length).

shear (image processing)

A skew is image distortion that often occurs when a scanner is sampling an image and the image slides to either side before the scan is complete. This has the effect of transforming squares into rhombuses.

special effects (image processing)

Any image processing transform that is applied mostly for its artistic value. Special effects include, wipes, transitions, barn doors, etc.

stretch intensity (image processing)

An image processing method that takes a given image and assures that the intensity distribution fills the entire range of possible values. An 8-bit image that is stretched always has at least one pixel with a value of zero and one of 255. The term comes from the before and after histogram of the given image. A stretch operation linearly stretches a histogram so that is ranges from the minimum pixel value to the maximum pixel value.

.TGA (file format extension)

Format originator:Truevision, Inc.

7340 Shadeland Station

Indianapolis, IN 46255

TIFF (file format)

Tagged **I**mage **F**ile **F**ormat.

.TIF (file format extension)

Format originator:Aldus Corp

411 First Ave South

Seattle, WA 98104, and

Microsoft Corp

16011 NE 36th Way

Redmond, WA 98073

thumbnail (imaging)

A small copy of an image. Thumbnails are used to display many images on the screen at once.

transform (image processing)

An algorithm that takes an image, alters it, and outputs a new image. Sometimes written as ‘xform’.

See Also:

- “geometric transform (image processing)” on page -567
- “neighborhood process (image processing)” on page -573
- “point process (image processing)” on page -574

triplet (digital imaging)

Three numbers used together to represent a single quantity or location, for example, RGB or (x, y, z).

(twenty-four) 24 bit image (digital imaging)

A 24-bit image contains pixels made from RGB triplets.

video stream (video)

A sequence of still images that are transmitted and displayed in synchronous order that give the appearance of live motion.

warp (image processing)

A geometric image processing routine that distorts an image by spatially compressing and stretching regions.

.WMF (file format extension)

Format originator:Microsoft Corp

16011 NE 36th Way

Redmond, WA 98073

world coordinates

The real valued coordinates that make sense for the object, treating it as if it really exists. The world coordinates of a house on a map would be in miles or longitude and latitude. This is the opposite of screen, device or model coordinates.

.WPG (file format extension)

Format originator:WordPerfect Corp

(x, y)

A mathematical method for referring to a pixel from a digital image. Since most digital images are maintained as a Cartesian matrix of pixels, each pixel has a unique address that can be described as an x or horizontal displacement from the origin and a y or vertical displacement from the origin.

See Also:

“coordinates” on page -563.

xform

Shorthand for transform.

YIQ (color imaging)

(**Y) luminance**, (**I**), (**Q**). YIQ is the color model used for U.S. commercial television. It was designed to be backwards compatible with the old black and white television sets. “Y” or luminance is a weighted average of the red, green, and blue that gives more weight to red and green than to blue. The I and Q contain the color components. Together they are called chromoticity.

(Z)

A mathematical method that refers to a pixel’s intensity from a digital image. An image can be written as: I(x,y)=z

Bibliography

Books

Brown, Wayne C., and Barry J. Shepherd. *Graphics File Formats Reference and Guide*. Greenwich, CT: Manning Publications, 1992.

Kay, David C. and John R. Levine. *Graphics File Formats*. Windcrest Books, 1992.

Kay, David C. and John R. Levine. *Graphics File Formats*. 2d ed. Windcrest/ McGraw-Hill, 1995.

Murray, James D., and William vanRyper. *Encyclopedia of Graphics File Formats*. Sebastapol, CA: O'Reilly & Associates, Inc., 1994.

Murray, James D. and William vanRyper. *Encyclopedia of Graphics File Formats,* 2d ed. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

Petzold, Charles. Programming Windows: The Microsoft Guide to Writing Applications for Windows 3. Redmond, WA: Microsoft Press, 1990.

Articles

Wegner, Tim. "Coding for PNG Graphics", *PC Techniques* (Feb/Mar 1996): 32-38.

Other

"Graphic Image Format FAQ 3-4." James D. Murray, 1994-1996.

PBM, PGM, PPM, PNM Specifications by Jef Poskanzer, copyright © 1989, 1991.

PNG (Portable Network Graphics), tenth draft. Page 5, copyright Thomas Boutell, May 1995.

Software License Agreement

YOUR INSTALLATION, USE AND DISTRIBUTION OF THIS SOFTWARE PRODUCT (“THE PRODUCT”) SIGNIFIES YOUR ACCEPTANCE OF THE TERMS OF THIS LICENSING AGREEMENT BETWEEN YOU (“LICENSEE”) AND ACCUSOFT® CORPORATION, A DELAWARE CORPORATION WITH ITS PRINCIPAL PLACE OF BUSINESS AT 71 LYMAN STREET, NORTHBOROUGH, MASSACHUSETTS 01532 (“ACCUSOFT”)

Article 1: Definitions.

Section 1.0. Defined Terms. Terms defined in this Article 1 shall have the meanings given below. Defined terms may be used in the singular or plural.

Section 1.1. “Agreement” means this Software License Agreement, which includes this Agreement’s terms and conditions, Order Forms, Schedules, Exhibits, Addenda, and Amendments, if any, which are incorporated in, and form an integral part of, this Agreement.

Section 1.2. “Application” means only the software application program or programs developed by Licensee (and all versions and releases thereof) that contain a copy of, and is linked to, the Product’s Distributable Files.

Section 1.3. “Business Day” means a day or part of a day when AccuSoft is open for business, excluding Saturdays, Sundays, official United States federal holidays, Commonwealth of Massachusetts state holidays, and AccuSoft holidays.

Section 1.4. “Confidential information” means any data or information, oral or written, that relates to either party’s (or, if either party is bound to protect the confidentiality of any other person's information, such other person's) past, present, or future research, development or business activities, including any unannounced product(s) and service(s), and including any information relating to services, developments, inventions, processes, plans, financial information, customer lists, forecasts, and projections, including the terms of this Agreement.

Section 1.5. “Develop” or “Development” means the development of one or more Applications by Licensee that contain a copy of the Product’s Distributable Files.

Section 1.6. “Distributable File” means those files contained in the Product that may be distributed with Licensee’s Application, as defined in Section 1.2. The list of Distributable Files for each Product is identified in the documentation that accompanies each Product.

Section 1.7. “Distribute” or “Distribution” means the distributing, deploying, selling, reselling, leasing, renting, licensing, lending or any other act whereby Licensee distributes the Application to an End User, whether such End User is internal or external to Licensee.

(a) “External Distribution” means that the Application containing the product is distributed for any reason other than internal distribution.

(b) “Internal Distribution” means that the Application containing the Product is not distributed for either commercial use or for sale (or resale), and is for use by End Users who are employees, interns, or contract employees of Licensee.

Section 1.8. “Distribution Method” means the method by which Licensee’s End User accesses the Distributable Files. A Distribution Method may include one or more of the following:

(a) “Client/Server” means the End User is permitted to install the Distributable File on a Server specifically designated to host the Application from which a fixed number of Named or Concurrent Users can access the Application at any one time.

(b) “Concurrent User” means the End User is permitted to have a fixed number of individual users who are authorized to use one copy of the Application on a single personal computer or a single networked computer at the same time.

(c) “Named User” means the End User is permitted to have only one single, named individual user who is authorized to use one copy of the Application on a single personal computer.

(d) “Server” means the End User is permitted to install the Distributable File on a networked processor specifically designated to process one or more instances of the Application (for example, batch imaging conversion processing, printer server processing, etc.).

Section 1.9. “End User” means the person or entity to which Licensee distributes the Application for use in the ordinary course of its business, and not for re-distribution or resale by such person or entity.

Section 1.10. “Installation” means the act of installing the Product on any system.

Section 1.11. “Licensing Fees” mean the fees identified in the Order Form that are mutually agreed to by AccuSoft and Licensee.

Section 1.12. “Order Form” means the order form, or other such document prescribed by AccuSoft, that details the Product(s), quantities, fees, and term of this Agreement and is executed by both parties.

Section 1.13. “Product” means the software product or products, including any accompanying documentation, identified in the Order Form and licensed by AccuSoft under this Agreement.

Section 1.14. “Supplier” means any software product or products, including any accompanying documentation, licensed by AccuSoft for use and distribution in the Product.

Section 1.15. “Term” means the period of time of this Agreement, as identified in the Order Form.

Article 2: Scope of License.

Section 2.1. Evaluation License. If the Product installed by Licensee is for evaluation purposes only, AccuSoft grants Licensee a license solely for the purpose of sampling and evaluating such Product for the timeframe specified in the evaluation period. No other rights, including but not limited to the rights stated in Sections 2.2 and 2.3, are applicable to evaluation copies of the product.

Section 2.2. Development License. Subject to the terms and conditions set forth in this Agreement, AccuSoft grants Licensee a non-exclusive, non-transferable, worldwide license to use the Product on a single machine, and incorporate all or portions of the Distributable Files into one or more Applications for Distribution by Licensee. Unless otherwise agreed to in writing by the parties, no more than one developer is authorized to use one single Product, and the Product is not to be used concurrently or on a site-wide basis, via a server or other networked connection. Licensee may make one copy of the Product for backup or archive purposes. If Licensee desires to have more than one developer use the Product, Licensee must purchase additional development licenses from AccuSoft.

Section 2.3. Distribution License. Subject to the terms and conditions set forth in this Agreement, AccuSoft grants Licensee a non-exclusive, non-transferable, worldwide license to distribute to End Users (either directly or indirectly) a copy of one or more distributable files in any applications developed pursuant to Section 2.2 by the Distribution Method identified on the Order Form during the Term of the Agreement. If Licensee distributes more licenses than it has pre-purchased, as identified on the Order Form, Licensee’s purchase of such additional licenses shall be subject to the terms and conditions set forth in Article 3 of this Agreement.

Section 2.4. Licensee shall distribute the Distributable Files in object code form only. No license or other right is granted to Licensee with respect to any of the Product’s source code. Licensee shall not alter, modify, disassemble, decompile or reverse engineer the Product in any way or manner.

Section 2.5. Licensee shall distribute the Application to its resellers, OEMs, or other distribution channels only pursuant to a written agreement that binds the resellers, OEMs, or other distribution channels to comply with the terms and conditions of Articles 2 and 5 of this Agreement.

Section 2.6. Licensee shall distribute the Application to its End Users only pursuant to a written agreement that prohibits the End User from Distributing, creating, copying, duplicating, re-distributing, disassembling, decompiling, reverse engineering, or sublicensing the Product, and shall use its best efforts to ensure the same.

Section 2.7. Licensee may distribute the Application only if the Application:

(a) operates solely on the hardware and operating system(s) for which the Product is intended to be used;

(b) does not extend access to the Product’s application programming interface (API);

(c) includes within the executable object files of all copies of the Product, including all Distributable Files, AccuSoft’s patent, copyright, trademark and proprietary notices; and

(d) recognizes AccuSoft’s copyright ownership in the Application initialization screen or “About Box”, and/or in the documentation for the Application as follows: *Imaging Technology provided under License by AccuSoft Corporation. ImageGear © 1996-2006. All Rights Reserved.”* Italics are not required. This requirement is not applicable if Licensee’s internal application has neither an initialization screen nor an “About Box.”

Section 2.8. If Licensee’s Application is intended for external distribution, Licensee’s Application shall also:

(a) not compete to any degree with the Product. Such competitive products are defined as imaging software development toolkit intended for use by application programmers and/or system integrators;

(b) be substantially greater in scope with greater functionality and features than those of the Product;

(c) not be a software development toolkit or provide a software development environment to the End User, and

(d) recognize AccuSoft’s copyright ownership in the Application initialization screen (or “About Box”) and/or in the documentation for the Application as follows: *Imaging Technology provided under License by AccuSoft Corporation. ImageGear © 1996-2006. All Rights Reserved.”* Italics are not required.

Section 2.9. As required by United States export regulations, Licensee shall not permit export of the Product, or any direct products thereof, to any country to which export is then controlled by the United States Bureau of Export Administration, unless Licensee has that agency's prior written approval. Licensee shall require its End Users to comply with the provisions of this Section.

Section 2.10. If Licensee (or Licensee's End User) is an office, branch or agency of the U.S. Government, then its use, duplication and disclosure of the Product or any part thereof, shall be subject to the following additional requirements: The Product is a "commercial item," as such term is defined in 48 C.F.R. 12.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F. 12.212 (September 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4, the U.S. Government End Users acquire the Product and/or Application with only those rights set forth in this section.

Section 2.11. AccuSoft reserves the right to include in the Product one or more software devices to ensure that Licensee's use of the Application is consistent with the use identified in the Order Form, and does not exceed the maximum number of fixed licenses granted by AccuSoft, if any.

Article 3: Fees and Reporting.

Section 3.1. Licensee shall pay to AccuSoft all Licensing Fees due for all copies of the Application containing the Distributable Files licensed and Distributed by Licensee. All Licensing Fees are payable in U.S. dollars and are non-refundable.

Section 3.2. Timely payment of the Licensing Fee permits Licensee to use the Product, and Distribute its Application (either directly or indirectly) to End Users. For U.S. based Licensees, payment is due 30 days from the date of the invoice. For all other Licensees, payment is due in full prior to or contemporaneously with the licensing of the Product. Late payment charges shall be imposed on all outstanding balances for each month or portion thereof that the outstanding balance remains unpaid, at the rate of 1.5% per month or the maximum rate permitted by law for such charge.

Section 3.3. Where required by law, AccuSoft will collect from Licensee, and Licensee is responsible to pay for, sales and all other taxes associated with the order, unless AccuSoft is provided a valid and correct tax exemption certificate prior to order acceptance. The payment of such tax is the sole responsibility of Licensee.

Section 3.4. Licensee understands and acknowledges that in order to receive bulk discount pricing for distribution Licensing Fees, it must pre-pay for such licenses. In the event that Licensee Distributes more licenses than it has purchased, or distributes by a Distribution Method not stated on the Order Form, Licensee shall pay AccuSoft's then-current quantity-one pricing for each copy distributed over the number licenses purchased. Quantity-one pricing is defined as the fee associated with purchasing one (1) distribution license.

Section 3.5. Licensee shall inform AccuSoft in writing within thirty (30) days after the date of the first shipment of copies of the Application. Thereafter, Licensee shall complete a License Distribution Report (or other such Distribution Fee statements provided by AccuSoft) each calendar quarter stating the number of Distribution Licenses distributed during each Disclosure Period and estimating the number of Distribution License that may be used during the next Disclosure Period. An authorized officer of Licensee shall sign all such License Distribution Reports.

Section 3.6. Upon AccuSoft's request, at mutually agreeable times no more frequently than once annually, an auditor chosen by AccuSoft shall be provided reasonable access during normal business hours to the relevant records of Licensee for the purpose of an audit of Distribution Fees due. Such records shall include, without limitation, information concerning copies of the Application made and distributed and other information necessary to verify the payments and amounts due AccuSoft hereunder. AccuSoft shall maintain in confidence any proprietary information of Licensee obtained during the course of such audit. The results of any such audit shall be final and dispositive. If the auditor's report indicates an underpayment of more than 10% for any auditing period, Licensee shall pay the full cost of the audit, and pay AccuSoft the balance due within thirty (30) days from presentation of the report.

Article 4: Support.

Section 4.1. For one year after Licensee's purchase or installation of the Product (pursuant to Section 2.2 of this Agreement), whichever is earlier, AccuSoft shall provide Licensee with a primary level of product support and maintenance, or a level and term agreed to by the parties and described on the Order Form. A copy of each of AccuSoft's product support and maintenance levels may be found in the on-line help section included with the Product.

Section 4.2. Licensee agrees that, within thirty days of its release of any Application built with the Product, it will provide AccuSoft with two (2) copies of such Application for support purposes. AccuSoft's use of such Application will be limited to providing support to Licensee, and shall be governed by the provisions of Article 7 of this Agreement.

Article 5: Intellectual Property.

Section 5.1. Licensee recognizes and acknowledges the exclusive right of AccuSoft (and its Suppliers) in and to all patents, trademarks, service marks, trade names, copyrights, and other intellectual property and proprietary rights in and to the Product, and that such patents, trademarks, service marks, trade names, copyrights, and other intellectual property and proprietary rights are the sole and exclusive property of AccuSoft and its Suppliers.

Section 5.2. Licensee waives its right to contest any of AccuSoft's or its Supplier's patents, trademarks, service marks, trade names, copyrights, and other intellectual property and proprietary rights in and to the Product.

Section 5.3. Licensee shall not use such trademarks, service marks, and trade names except where permitted under this Agreement without receiving AccuSoft's prior written approval of such use. If such approval is granted, Licensee's right to use such trademarks, service marks, and trade names shall end upon the termination of this Agreement.

Article 6: Confidentiality.

Section 6.1. Licensee shall remain in exclusive control of the Product, including but not limited to, the physical media, downloads, sample code, and documentation. The parties agree that neither party shall disclose or otherwise make available any confidential information provided to it by the other party. Unless otherwise agreed to by the parties, the scope of this Section shall be given the broadest interpretation possible and shall include, but not be limited to, the terms and conditions of this Agreement, the Order Form, and pricing.

Section 6.2. The obligations of non-disclosure in Section 6.1 shall not apply to information that (i) is publicly available or in the public domain at the time disclosed; (ii) is, or becomes, publicly available or enters the public domain through no fault of the party receiving such information; (iii) is rightfully communicated to the recipient by persons not bound by confidentiality obligations with respect thereto; (iv) is already in the recipient's possession free of any confidentiality obligations with respect thereto at the time of disclosure; (v) is independently developed by the recipient; (vi) is approved for release or disclosure by the disclosing party without restriction, (vii) is required by a court of competent jurisdiction.

Article 7: Term and Termination.

Section 7.1. Unless otherwise agreed to by the parties, this Agreement shall become effective upon its execution by both parties, and shall continue in full force and effect until terminated in accordance with the terms set forth in the Order Form or this Agreement.

Section 7.2. In the event that Licensee materially breaches this Agreement, or ceases business or is adjudged bankrupt or insolvent, then AccuSoft may, at its option, terminate this Agreement, by giving ten (10) Business Days written notice of such termination, which notice shall identify and describe the basis for such termination. If, prior to expiration of such period, Licensee cures such default, termination shall not take place.

Section 7.3. In the event of any termination of this Agreement, Licensee shall stop manufacturing and Distributing the Application containing a copy of the Distributable Files, and require its resellers, OEMs, and other distribution channels (if any) to likewise stop manufacturing and Distributing the Application containing a copy of the Distributable Files. Within ten (10) Business Days thereafter, Licensee shall return or, at AccuSoft's option, destroy the Product and all copies in whole or in part thereof, whether or not incorporated in or with the Application, and within Licensee's possession, custody and control, and shall certify to AccuSoft in writing within ten (10) Business Days thereafter that it has complied with the foregoing obligation.

Section 7.4. For external distributions, the termination of this Agreement shall not affect any rights of End Users to use the Application if such copies were shipped prior to the termination of this Agreement. So long as Licensee's external distributions to End Users and the End Users' use of the Application conforms to the terms and conditions set forth in this Agreement, the End User's rights to use the Product as incorporated into the Application shall be perpetual.

Section 7.5. Articles 1, 5, 6, 7, 8 and 9 shall continue in full force and effect, notwithstanding any such termination of this Agreement.

Article 8: Limited Warranty.

Section 8.1. AccuSoft represents and warrants that it has full power and authority to enter into this Agreement, and has sufficient right, title, and interest in the Product to enter into this Agreement.

Section 8.2. Licensee understands and acknowledges that the Product, although functionally complete, has not been extensively tested by AccuSoft or its Suppliers as to the correctness or speed of operation in Licensee's Application. Although AccuSoft will make reasonable efforts to correct any material nonconformity between the Product and its then-current documentation that is brought to AccuSoft's attention with written, documented reports from Licensee, ACCUSOFT AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE RESULTS OBTAINED BY USING THE PRODUCT. AccuSoft's entire liability and Licensee's exclusive remedy under this Agreement shall be to provide reasonable support services.

Section 8.3. THE PRODUCT IS PROVIDED “AS IS.” ACCUSOFT AND ITS SUPPLIERS SPECIFICALLY DISCLAIM ANY AND ALL WARRANTY TO LICENSEE OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, PERFORMANCE RESULTS AND NON INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PRODUCT IS WITH LICENSEE.

Section 8.4. IN NO EVENT WILL ACCUSOFT OR ITS SUPPLIERS BE LIABLE TO LICENSEE FOR ANY DAMAGES, INCLUDING LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OR INABILITY TO USE THE PRODUCT EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to Licensee. To the extent permissible, any implied warranties are limited to thirty (30) days. This warranty gives Licensee specific legal rights. Licensee may have other rights which vary from state to state or jurisdiction to jurisdiction.

Section 8.5. The warranty extended in this Article extends only to Licensee, and shall not apply to any other person or entity including, but not limited to, Licensee’s End Users.

Article 9: General Provisions.

Section 9.1. AccuSoft and Licensee warrant that they have the requisite authority to enter into this Agreement.

Section 9.2. This Agreement constitutes the entire understanding and agreement between AccuSoft and Licensee regarding its subject matter, and supercedes all previous oral and written communications, agreements, memoranda, representations, or understandings between AccuSoft and Licensee regarding this Agreement. No other rights or licenses are granted to Licensee, except as expressly provided herein.

Section 9.3. This Agreement shall not be amended, altered, changed or modified in any way, unless agreed to in writing by both AccuSoft and Licensee. Such writing must be executed by a duly authorized representative of AccuSoft and a duly authorized representative of Licensee.

Section 9.4. This Agreement is not transferable or assignable by Licensee under any circumstances, without the prior written consent of AccuSoft. AccuSoft will not unreasonably withhold such consent. This agreement shall be binding upon, and is made for the benefit of, each party, its successors, and permitted assignees (if any). For the purposes of this Agreement, any change in control of Licensee shall constitute an assignment or transfer of this Agreement. As used in this section, a change in control is defined as (i) any change in ownership of more than fifty percent (50%) of the voting interest in Licensee, whether by merger, purchase, foreclosure of a security interest or other transaction, or (ii) a sale of all or substantially all of the assets of Licensee.

Section 9.5. The relationship established by this Agreement between Licensee and AccuSoft shall be that of licensee and licensor. Nothing contained in this Agreement shall be construed as creating a relationship of agency, joint venture or partnership between Licensee and AccuSoft. Neither party shall have any right whatsoever to incur any liabilities or obligations on behalf of the other party.

Section 9.6. The laws of the Commonwealth of Massachusetts shall govern this Agreement, excluding that body of law known as conflict of laws and the United Nations Convention on Contracts for the Sale of Goods. Any action or proceeding brought by either party against the other arising out of, or related to, this Agreement shall be brought only in a state or federal court of competent jurisdiction located in the Commonwealth of Massachusetts and the parties hereby consent to the personal jurisdiction of said courts.

Section 9.7. In the event that any provision of this Agreement is found invalid or unenforceable pursuant to a judicial decree or decision, the remainder of this Agreement shall remain valid and enforceable according to its terms.

Section 9.8. AccuSoft’s failure to perform any term or condition of this Agreement as a result of conditions beyond its control such as, but not limited to, war, strikes, fires, floods, acts of God, governmental restrictions, power failures, or damage or destruction of any network facilities or servers, shall not be deemed a breach of this Agreement.

Section 9.9. In the event of suit or any legal action due to any breach of this Agreement is commenced to enforce this Agreement or otherwise relating to this Agreement, the prevailing party shall be entitled to all attorneys’ fees and costs incurred in connection therewith.

Section 9.10. The headings provided in this Agreement are for convenience and reference purposes only. In the event of a conflict between the terms and conditions listed in Articles 1 through 10, and the attached Schedules, the terms and conditions shall govern.

Section 9.11. A waiver of a breach, violation, or default under this Agreement shall not be a waiver of any subsequent breach, violation or default. Failure of either party to enforce compliance with any term or condition of this Agreement shall not constitute a waiver of such term or condition.

Section 9.12. All notices and communications shall be in writing and shall be deemed to have been duly given when delivered or three (3) Business Days after mailing by certified mail, return receipt requested, postage prepaid, addressed to the parties at their respective addresses set forth on the Order Form or at such other addresses as the parties may designate by written notice in accordance with this section.

Section 9.13. The Order Form, and any amendments or addenda to this Agreement, may be executed in counterparts, each of which will be considered an original, but all counterparts together will constitute one agreement. A facsimile of a signed copy of this Agreement, or an electronic or other digital signature imprinted on this Agreement, may be relied upon as an original.

Schedule A: Additional Terms and Conditions Applicable to Adobe® Technology

1. Adobe Technology

If Licensee's use and distribution of the Product includes Adobe Technology and related documentation, the term "Product" as defined in Section 1.13 of the Agreement includes Adobe Technology and related documentation, and any upgrades, modified versions, updates, additions, and copies thereof.

2. Font License

If the Product includes font software, Licensee may embed the font software, or outlines of the font software, into its Application to the extent that the font vendor copyright owner allows for such embedding. The fonts contained in this package may contain both Adobe and non-Adobe owned fonts. Licensee may fully embed any font owned by Adobe.

3. Limited Warranty

ARTICLE 8 OF THE AGREEMENT STATES THE SOLE AND EXCLUSIVE REMEDIES FOR ACCUSOFT'S BREACH OF WARRANTY. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, ADOBE AND ITS SUPPLIERS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AS TO MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL ADOBE OR ITS SUPPLIERS BE LIABLE TO LICENSEE FOR ANY CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, EVEN IF AN ADOBE REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to Licensee. To the extent permissible, any implied warranties are limited to thirty (30) days. This warranty gives Licensee specific legal rights. Licensee may have other rights which vary from state to state or jurisdiction to jurisdiction.

4. Trademarks

Adobe and Adobe PDF Library are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Schedule B: Additional Terms and Conditions Applicable to MIT's FFTW Technology

1.FFTW Technology

If Licensee's use and distribution of the Product includes FFTW Technology, please note that the FFTW technology is provided to AccuSoft under a copyright license from the Massachusetts Institute of Technology subject to the following terms:

2.Sublicensing

If Licensee is not the End User of the FFTW Technology and intends to distribute the Product (including the FFTW Technology) to the End User, AccuSoft is required notify M.I.T. each calendar quarter of this licensing agreement, including the name and contact information of Licensee. M.I.T. shall treat these sublicenses as confidential information of the COMPANY.

3.Termination

In the event MIT terminates its FFTW copyright license with AccuSoft, Licensee's rights to use the FFTW technology may be affected as follows:

a. If Licensee is an End User of the Product, and has no rights to sublicense or distribute the Product, Licensee's right to continued use of the FFTW technology within the Product shall not be affected by such termination.

b. If Licensee is not an End User of the Product, and has the rights to sublicense or distribute the Product containing the FFTW technology, Licensee's rights to use the FFTW technology shall terminate. Upon such termination, Licensee has the right to see a license directly from M.I.T. M.I.T. agrees to negotiate such licenses in good faith under reasonable terms and conditions.

4.Retained Rights

a. Non-Commercial Use. Regardless of any intellectual property rights to be established by AccuSoft, M.I.T. hereby retains now and forever the right to modify the FFTW Technology in any way whatsoever, the right to use the FFTW Technology for M.I.T.'s educational and research purposes, the right to distribute the FFTW Technology to the United States government or to any United States government contractor for use on a United States government contract, and to distribute the FFTW Technology and M.I.T.-created modifications, improvements, enhancements, or upgrades to third parties for educational, non-profit and non-commercial use.

b. Federal Government. AccuSoft acknowledges that the U.S. federal government retains a royalty-free, non-exclusive, non-transferable license to the FFTW Technology pursuant to 48 CFR 52.227-14 (Civilian Agencies) or DFARS 252.227-7014 (Defense Agencies).

c. Limitation. Nothing in this Agreement shall be construed to confer any rights upon AccuSoft by implication, estoppel or otherwise to any computer software, trademark, technology or patent rights of M.I.T., or of any other entity, except as explicitly granted. Licensee acknowledges that this Agreement does not transfer title to FFTW Technology to AccuSoft or Licensee. Nothing in this Agreement shall be construed as a transfer of title to the FFTW Technology to AccuSoft or Licensee by implication, estoppel or otherwise. To the extent the FFTW Technology contains a copyright notice, AccuSoft and Licensee shall retain prominently such copyright notice.

d. Disclaims of Liability

ARTICLE 8 OF THE AGREEMENT STATES THE SOLE AND EXCLUSIVE REMEDIES FOR ACCUSOFT'S BREACH OF WARRANTY. THE FFTW TECHNOLOGY IS DELIVERED "AS IS." M.I.T. MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE FFTW TECHNOLOGY OR THE COPYRIGHT, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, WHETHER OR NOT DISCOVERABLE. M.I.T. EXTENDS NO WARRANTIES OF ANY KIND AS TO THE FFTW TECHNOLOGY'S CONFORMITY WITH WHATEVER USER MANUALS OR OTHER LITERATURE MAY BE ISSUED FROM TIME TO TIME. Furthermore, and not to limit the foregoing, M.I.T. makes no warranty or representation that the exploitation of the FFTW technology or any derivative will not infringe any patents or other intellectual property rights of M.I.T. or of a third party.

IN NO EVENT SHALL M.I.T., ITS TRUSTEES, DIRECTORS, OFFICERS, EMPLOYEES AND AFFILIATES BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY KIND, INCLUDING ECONOMIC DAMAGES OR INJURY TO PROPERTY AND LOST PROFITS, REGARDLESS OF WHETHER M.I.T. SHALL BE ADVISED, SHALL HAVE OTHER REASON TO KNOW, OR IN FACT SHALL KNOW OF THE POSSIBILITY OF THE FOREGOING.

Software Support and Maintenance Policy

AccuSoft provides product support and maintenance services for all of its products. These services provide the subscriber with a certain level of ongoing technical assistance (i.e., support) and access to software updates (i.e., maintenance). The subscriber is the individual or company who has licensed one or more products from AccuSoft, and who is entitled to product support and maintenance services under the terms and conditions of its Software License Agreement with AccuSoft. Additionally, unless otherwise stated in this policy, the terms used in this policy shall have the same meanings as the terms defined in Article I of the Software License Agreement.

See Also:

- Primary Level
- Preferred Level
- Premier Level

Primary Level

At the Primary Level, product support and maintenance services entitle a subscriber to the following:

1. Maintenance.

AccuSoft makes periodic updates to its products to correct errors, make enhancements, or improve performance. An update is defined as a minor release to the current version that results in a patch, bug fix, updated documentation, or improved performance of the Product. At the Primary Level, subscribers are entitled to any and all updates to the version of the product licensed by the subscriber during the support term.

2. Technical Assistance.

Technical assistance is provided to subscribers to enable AccuSoft to determine and resolve product problems and questions. Technical assistance does not include software application consulting, or custom coding of a subscriber's solution.

Technical assistance is only provided via electronic methods from the AccuSoft support web site to the individual identified by the subscriber as the designated support contact for the product.

Primary support provides problem ticketing and resolution via a web-interface only. During the support term, the designated support contact will have unlimited access to the AccuSoft web site for ticket creation, status monitoring, information access, etc.

Primary Support services include:

- Electronic access to AccuSoft's support forums
- Electronic submission of product problems and questions, and online access to update and track the progress of your case
- Electronic access to up-to-date knowledge databases for our products that provide answers and fixes to commonly asked questions and problems
- Electronic access to product documentation including the updated guides and manuals, technical bulletins, flashes, and release notes
- Electronic access to all patches, product fixes, release and maintenance information for the version licensed

3. Support Services Availability and Coverage.

Electronic access is available 24 hours a day, 7 days a week.

4. Support Services – Response Objectives.

Primary Support does not provide a guaranteed personal response to an issue, but all issues logged will be reviewed and evaluated as to their suitability for posting as a resolution for the AccuSoft community as a whole.

5. Additional Support Options.

During the first ninety (90) days after the purchase of the product, a subscriber is eligible to upgrade to an enhanced level of support services that includes but is not limited to, version upgrades, guaranteed response times and access to AccuSoft's Customer Support Department.

Additionally, AccuSoft may also provide on-site technical assistance to augment any such remote technical assistance. On-site technical assistance shall be chargeable to the subscriber based on AccuSoft's then current on-site support services fees and policies. On-site technical assistance is not included in the services provided in the maintenance and support subscription described above.

6. Subscriber Responsibilities.

- **Information.** Subscriber problem reports to AccuSoft must contain all pertinent information regarding the issue, including but not limited to, the following: the AccuSoft software/revision number, Operating system/ version, Development Platform, Problem Description, Log files/Test Case and Problem Severity. Subscribers must execute diagnostic routines if provided by AccuSoft, and inform AccuSoft of the results. Subscribers must also verify the existence of a product problem and provide the conditions under which it may be duplicated by AccuSoft at an AccuSoft location. All problem reports must be in English.
- **Technical Contact.** During the period of time in which AccuSoft is working on the subscriber issue, the subscriber shall provide a technical contact who is familiar with the product problem and the subscriber environment.

7. Service Limitations.

- **AccuSoft will support only the most current version of the Product.** The current version is defined as the version of the product that is commercially available for sale by AccuSoft. Upon the release of such current version of a product, AccuSoft will continue to support and maintain the version of the product released immediately prior to the release of the current version in accordance with the then prevailing End-of-Life policy.
- **All associated computer hardware and operating system software and product must be maintained at the latest version deemed necessary by AccuSoft.** AccuSoft shall not be obligated to provide support services under this policy if the service is regarding hardware, software or product that does not conform to the above statement. Any and each such service may result in additional charges for labor and materials as determined by AccuSoft, and shall be provided at the sole discretion of AccuSoft.
- **On-site services will be provided by quotation, are not covered by this policy, and shall be provided at the sole discretion of AccuSoft.** Additional AccuSoft consulting and/or coding services may also be available to the subscriber under a separate agreement. AccuSoft may provide assistance to help optimize or enhance subscriber's use of the product. When AccuSoft determines that any such assistance has exceeded a reasonable level, AccuSoft will supply a quotation for additional Consulting Services.
- **Primary Level subscribers will receive any software updates AccuSoft makes generally available during the maintenance term.** Primary Level subscribers, however, are not entitled to receive any upgrades or significant new enhanced functionality. An upgrade is defined as a change to the product that substantially enhances the functionality of the product and/or results in change in the version number of the product to the left of the decimal point.

See Also:

- Preferred Level
- Premier Level

Preferred Level

At the Preferred Level, product support and maintenance services entitle a subscriber to the following:

1. Maintenance.

AccuSoft makes periodic updates to its products to correct errors, make enhancements, or improve performance. An update is defined as a minor release to the current version that results in a patch, bug fix, updated documentation, or improved performance of the Product. At the Preferred Level, subscribers are entitled to any and all updates to the version of the product licensed by the subscriber during the support term.

At the Preferred Level, subscribers are also entitled to any and all upgrades to the version of the product licensed by the subscriber during the support term. An upgrade is defined as a change to the product that substantially enhances the functionality of the product and/or results in change in the version number of the product to the left of the decimal point.

2. Technical Assistance.

Technical assistance is provided to subscribers to enable AccuSoft to determine and resolve product problems and questions. Technical assistance does not include software application consulting, or custom coding of a subscriber's solution.

Technical assistance is only provided via electronic methods from the AccuSoft support web site to the individual identified by the subscriber as the designated support contact for the product.

Preferred support is entirely web-based for both problem ticketing and resolution. During the support term, Preferred Level subscribers receive automated e-mail notification of any updates to ticket that have been posted to AccuSoft's support web site. The designated support contact will have unlimited access to the AccuSoft web site for ticket creation, status monitoring, information access, etc.

Preferred Support services include:

- Electronic access to AccuSoft's support forums
- Electronic submission of product problems and questions, and online access to update and track the progress of your case
- Electronic access to up-to-date knowledge databases for our products that provide answers and fixes to commonly asked questions and problems
- Electronic access to product documentation including the updated guides and manuals, technical bulletins, flashes, and release notes
- Electronic access to all patches, product fixes, release and maintenance information for the version licensed

Plus,

- Electronic access to all updates, enhancements and upgrades (including new versions/releases) that may be issued to the licensed product during your support term (one-year by default)
- Electronic notification that a resolution or status update is available to view through AccuSoft's electronic web interface

3. Support Services Availability and Coverage.

Electronic access is available 24 hours a day, 7 days a week.

AccuSoft's Customer Support department is open Monday through Friday, 8:30 a.m. to 5:30 p.m. EST, except on designated AccuSoft holidays. Requests for assistance received after the daily coverage period shall be deemed to have been received the next business day.

4. Support Services – Response Objectives.

AccuSoft shall use reasonable efforts to meet the following objectives with regard to remedial support for each AccuSoft productsoftware.

- **Initial Response.** Upon receipt, AccuSoft will respond to the subscriber's report of a problem with the product based upon the initial response time and problem severity priority listed below:

Problem Resolution Stages for Preferred Level	**Problem Severity**		
	Priority I	**Priority II**	**Priority III**
Initial Response	3 Business Hours	8 Business Hours	24 Business Hours
Response Update	Every 2 Business Days	Every 2 Weeks	Every 4 Weeks
Relief	Priority work	ASAP	Reasonable efforts

- **Problem Severity.** Problem Severity is prioritized as follows:

  Priority I – Subscriber is unable to do production work or production is jeopardized and a work-around is not available or is generally unacceptable;

  Priority II – A major product function is inoperable, but production can continue;

  Priority III – A loss of a function that does not seriously impact operation or schedules.

- **Problem Resolution Stages.** After the initial response, AccuSoft will provide a response update to the subscriber according to the schedule specified above. AccuSoft will use the following objectives for problem resolution: Response – A technically qualified engineer is working on the problem; Relief – subscriber's operation is no longer at risk due to the original problem, or the subscriber understands and accepts all risk. A permanent problem resolution may be available in a subsequent revision.

5. Additional Support Options.

During the first ninety (90) days after the purchase of the product, a subscriber is eligible to upgrade to an enhanced level of support services that includes but is not limited to, access to AccuSoft's Customer Support Department.

Additionally, AccuSoft may also provide on-site technical assistance to augment any such remote technical assistance. On-site technical assistance shall be chargeable to the subscriber based on AccuSoft's then current on-site support services fees and policies. On-site technical assistance is not included in the services provided in the maintenance and support subscription described above.

6. Subscriber Responsibilities.

- **Information.** Subscriber problem reports to AccuSoft must contain all pertinent information regarding the issue, including but not limited to, the following: the AccuSoft software/revision number, Operating system/ version, Development Platform, Problem Description, Log files/Test Case and Problem Severity. Subscribers must execute diagnostic routines if provided by AccuSoft, and inform AccuSoft of the results. Subscribers must also verify the existence of a product problem and provide the conditions under which it may be duplicated by AccuSoft at an AccuSoft location. All problem reports must be in English.
- **Technical Contact.** During the period of time in which AccuSoft is working on the subscriber issue, the subscriber shall provide a technical contact who is familiar with the product problem and the subscriber environment.
- **Remote Access.** By purchasing software support and maintenance services, the subscriber agrees to permit AccuSoft to remotely access the subscribers computer system for the sole purpose of diagnosing any reported problem with the product including, but not limited to, granting AccuSoft permission to install remote access software on subscriber's computer system(s) that contain the product. All standard confidentiality provisions identified in the Software License Agreement will apply. The subscriber shall provide remote access via dial-in or internet when required by AccuSoft to analyze a problem.

7. Service Limitations.

- **AccuSoft will support only the most current version of the Product.** The current version is defined as the version of the product that is commercially available for sale by AccuSoft. Upon the release of such current version of a product, AccuSoft will continue to support and maintain the version of the product released immediately prior to the release of the current version in accordance with the then prevailing End-of-Life policy.
- **All associated computer hardware and operating system software and product must be maintained at the latest version deemed necessary by AccuSoft**. AccuSoft shall not be obligated to provide support services under this policy if the service is regarding hardware, software or product that does not conform to the above statement. Any and each such service may result in additional charges for labor and materials as determined by AccuSoft, and shall be provided at the sole discretion of AccuSoft.
- **On-site services will be provided by quotation, are not covered by this policy, and shall be provided at the sole discretion of AccuSoft.** Additional AccuSoft consulting and/or coding services may also be available to the subscriber under a separate agreement. AccuSoft may provide assistance to help optimize or enhance subscriber's use of the product. When AccuSoft determines that any such assistance has exceeded a reasonable level, AccuSoft will supply a quotation for additional Consulting Services.

See Also:

- Primary Level
- Premier Level

Premier Level

At the Premier Level, product support and maintenance services entitle a subscriber to the following:

1. Maintenance.

AccuSoft makes periodic updates to its products to correct errors, make enhancements, or improve performance. An update is defined as a minor release to the current version that results in a patch, bug fix, updated documentation, or improved performance of the Product. At the Premier Level, subscribers are entitled to any and all updates to the version of the product licensed by the subscriber during the support term.

At the Premier Level, subscribers are also entitled to any and all upgrades to the version of the product licensed by the subscriber during the support term. An upgrade is defined as a change to the product that substantially enhances the functionality of the product and/or results in change in the version number of the product to the left of the decimal point.

2. Technical Assistance.

Technical assistance is provided to subscribers to enable AccuSoft to determine and resolve product problems and questions. Technical assistance does not include software application consulting, or custom coding of a subscriber's solution.

Premier support is pro-active, with personalized contact both to and from the subscriber via any and all contact methods – e-mail, telephone, facsimile – as well as AccuSoft's electronic web interface. The designated support contact may make unlimited requests for assistance to AccuSoft's Customer Service department.

Premier Support services include:

- Electronic access to AccuSoft's support forums
- Electronic and/or personalized submission of product problems and questions, and online access to update and track the progress of your case
- Electronic access to up-to-date knowledge databases for our products that provide answers and fixes to commonly asked questions and problems
- Electronic access to product documentation including the updated guides and manuals, technical bulletins, flashes, and release notes
- Electronic access to all patches, product fixes, release and maintenance information for the version licensed

Plus,

- Electronic access to all updates, enhancements and upgrades (including new versions/releases) that may be issued to the licensed product during your support term (one-year by default)
- Electronic notification that a resolution or status update is available to view through AccuSoft's electronic web interface

Additionally, should AccuSoft's Customer Support analysts need to investigate an issue within your own computer environment, the Customer Support analysts may recommend the use of our e-support solution. This system allows AccuSoft to assist the subscriber by allowing the analyst to:

- Access your system to view, diagnose and solve your problems online
- Run applications to ensure proper installation and configuration
- Download and install patches or updates to your system
- Upload your files for analysis
- Include other analysts for consultation
- Interact with you via text chat
- Record the entire support session for archival

3. Support Services Availability and Coverage.

Electronic access is available 24 hours a day, 7 days a week.

AccuSoft's Customer Support department is open Monday through Friday, 8:30 a.m. to 5:30 p.m. EST, except on designated AccuSoft holidays. Requests for assistance received after the daily coverage period shall be deemed to have been received the next business day.

4. Support Services – Response Objectives.

AccuSoft shall use reasonable efforts to meet the following objectives with regard to remedial support for each AccuSoft productsoftware.

- **Initial Response.** Upon receipt, AccuSoft will respond to the subscriber's report of a problem with the product based upon the initial response time and problem severity priority listed below:

Problem Resolution Stages for Premier Level	**Problem Severity**		
	Priority I	**Priority II**	**Priority III**
Initial Response	1 Business Hour	4 Business Hours	12 Business Hours

Response Update	Each Business Day	Every Week	Every 2 Weeks
Relief	Work Continuously	ASAP	Reasonable efforts

- **Problem Severity**. Problem Severity is prioritized as follows:

  Priority I – Subscriber is unable to do production work or production is jeopardized and a work-around is not available or is generally unacceptable;

  Priority II – A major product function is inoperable, but production can continue;

  Priority III – A loss of a function that does not seriously impact operation or schedules.

- **Problem Resolution Stages.** After the initial response, AccuSoft will provide a response update to the subscriber according to the schedule specified above. AccuSoft will use the following objectives for problem resolution: Response – A technically qualified engineer is working on the problem; Relief – subscriber's operation is no longer at risk due to the original problem, or the subscriber understands and accepts all risk. A permanent problem resolution may be available in a subsequent revision.

5. Additional Support Options.

AccuSoft may also provide on-site technical assistance to augment any such remote technical assistance. On-site technical assistance shall be chargeable to the subscriber based on AccuSoft's then current on-site support services fees and policies. On-site technical assistance is not included in the services provided in the maintenance and support subscription described above.

6. Subscriber Responsibilities.

- **Information.** Subscriber problem reports to AccuSoft must contain all pertinent information regarding the issue, including but not limited to, the following: the AccuSoft software/revision number, Operating system/ version, Development Platform, Problem Description, Log files/Test Case and Problem Severity. Subscribers must execute diagnostic routines if provided by AccuSoft, and inform AccuSoft of the results. Subscribers must also verify the existence of a product problem and provide the conditions under which it may be duplicated by AccuSoft at an AccuSoft location. All problem reports must be in English.

- **Technical Contact.** During the period of time in which AccuSoft is working on the subscriber issue, the subscriber shall provide a technical contact who is familiar with the product problem and the subscriber environment.

- **Remote Access.** By purchasing software support and maintenance services, the subscriber agrees to permit AccuSoft to remotely access the subscriber's computer system for the sole purpose of diagnosing any reported problem with the product including, but not limited to, granting AccuSoft permission to install remote access software on subscriber's computer system(s) upon which the product has been installed. All standard confidentiality provisions identified in the Software License Agreement will apply. The subscriber shall provide remote access via dial-in or internet when required by AccuSoft to analyze a problem.

7. Service Limitations.

- **AccuSoft will support only the most current version of the Product.** The current version is defined as the version of the product that is commercially available for sale by AccuSoft. Upon the release of such current version of a product, AccuSoft will continue to support and maintain the version of the product released immediately prior to the release of the current version in accordance with the then prevailing End-of-Life policy.

- **All associated computer hardware and operating system software and product must be maintained at the latest version deemed necessary by AccuSoft.** AccuSoft shall not be obligated to provide support services under this policy if the service is regarding hardware, software or product that does not conform to the above statement. Any and each such service may result in additional charges for labor and materials as determined by AccuSoft, and shall be provided at the sole discretion of AccuSoft.

- **On-site services will be provided by quotation, are not covered by this policy, and shall be provided at the sole discretion of AccuSoft.** Additional AccuSoft consulting and/or coding services may also be available to the subscriber under a separate agreement. AccuSoft may provide assistance to help optimize or enhance subscriber's use of the product. When AccuSoft determines that any such assistance has exceeded a reasonable level, AccuSoft will supply a quotation for additional Consulting Services.

See Also:

- Primary Level
- Preferred Level

Volume 2: API Reference

This part of ImageGear User's Manual provides detailed information about each function or control parameter of the ImageGear Components. This information includes function's calling sequence, arguments, use, possible return values, and supported bit depths as well as other useful information.

Currently there are the following components API reference:

- Core Component API Function Reference
- ART Component API Function Reference
- MD Component API Function Reference
- PDF Component API Function Reference

Core Component API Function Reference

In this Chapter you can get the detailed information about all ImageGear Core component functions and callbacks:

- Core Component Functions Reference
- Core Component Callback Functions Reference

Core Component Functions Reference

This section describes each function supported by ImageGear Core component. They are presented in alphabetical order.

IG_ascii_import

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_ascii_import (const LPSTR lpszFileName,
   UINT nPageNumber,
   LPHIGEAR lphIGear
   );
```

Arguments:

lpszFileName	Set to the path/filename of the text file to load.
nPageNumber	Set to the number of the page to load; if not a multipage file, set to 1.
lphIGear	A far pointer that returns a `HIGEAR` handle for your newly loaded image.

Description:

This function loads an ASCII (.TXT) file into ImageGear. Specify the path/filename of the file and which page of the file (if it is a multi-page file) you would like to load. ImageGear then returns a **`HIGEAR`** handle to the newly loaded image.

To set the control parameters of the loaded file, use **`IG_fltr_ctrl_set()`**. The TXT filter control parameters are listed in the section TXT (ASCII Text).

You may also use **`IG_load_file()`** to load an ASCII file.

IG_ascii_page_width_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_ascii_page_width_get (const LPSTR
lpszFileName, LPUINT lpPageWidth);
```

Arguments:

lpszFileName	Set to the name of the ASCII file from which to get the width.
lpPageWidth	A far pointer that returns a value of type `UINT` indicating the width of the file in thousands of an inch.

Description:

This function returns the width of an ASCII file that has not been loaded yet. Simply supply it with the path/filename of an ASCII file and it will return the width in thousands of an inch. For example, if the file has a width of 8.5 inches, this function returns the value 8500.

The width of the page (as well as many other attributes) can be set before the page is loaded. See the description for **`IG_ascii_import()`** for the full list of attributes that can be determined prior to loading.

As an alternative to this function, you can use **`IG_fltr_ctrl_get()`** with the control parameter argument "`PAGE_WIDTH`".

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image   */
AT_ERRCOUNT nErrcount;    /* total # of ImageGear errors on the
stack*/
UINT pageWidth;     /* returns the width of the ASCII file */
UINT pointSize;    /* to get and possibly set the point size of
the font */
AT_REAL reduction;   /* percentage of reduction possibly needed
on the point size before the image is loaded with a width of 8.5
inches   */

nErrcount = IG_ascii_page_width_get("Hamlet.txt", &pageWidth);
if (pageWidth > 8500)
```

```
/* if the width of the unloaded page is greater than 8.5 inches,
we will resize it to 8.5 inches before loading it. However, to
avoid cropping any words, we will first reduce the point size by
the percentage needed to make each line fit on an 8.5 inch page
*/
{
    reduction = 8500/pageWidth;
    pointSize = (UINT)(pointSize * reduction + 0.5);
    nErrcount =
        IG_fltr_ctrl_set(IG_FORMAT_TXT, “POINT_SIZE”,
        (LPVOID)pointSize, sizeof(pointSize));
    nErrcount =
        IG_fltr_ctrl_set(IG_FORMAT_TXT, “PAGE_WIDTH”,
        (LPVOID)8500, sizeof(nPageWidth));
}
nErrcount = IG_ascii_import("Hamlet.txt", 1, &hIGear);
```

IG_clipboard_copy

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_copy ( HIGEAR hIGear,
    const LPAT_RECT lprcRectToCopy
    );
```

Arguments:

hIGear	HIGEAR handle of the current image.
rcRectToCopy	Far pointer to an AT_RECT struct specifying the rectangular portion of the image that is to be copied to the clipboard.

Description:

This function copies the specified portion of the image to the system clipboard. If rcRectToCopy = NULL, the entire image will be copied.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR       hIGear;        /* HIGEAR handle of image  */
AT_RECT      rcRectToCopy;  /* Rectangle of image to copy to
clipboard */
AT_DIMENSION nImageWidth;    /* Width of image  */
AT_DIMENSION nImageHeight;  /* Height of image  */

/* Copy bottom half of image to system clipboard:  */
rcRectToCopy.left   = 0;
rcRectToCopy.top    = nImageHeight / 2;
rcRectToCopy.right  = nImageWidth - 1;
rcRectToCopy.bottom = nImageHeight - 1;
IG_clipboard_copy ( hIGear, &rcRectToCopy );
```

IG_clipboard_cut

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_cut (HIGEAR hIGear, const
LPAT_RECT lprcRegion, const LPAT_PIXEL lpPixel);
```

Arguments:

hIGear	HIGEAR handle of current image
lprcRegion	A long pointer to a rectangular region of the image, which should be copied to the clipboard and removed from the image.
lpPixel	A long pointer to a pixel value you would like to be used to fill in the area out of which the region has been cut.

Description:

This function “cuts away” a portion of an image. The cut portion is copied to the clipboard with its original pixel values, while in the displayed image, that rectangle is replaced by a pixel value as specified by **lpPixel**. The color used is usually

black or white. You can restore the image to its original composition by calling **IG_clipboard_paste_merge_ex()**. You then set the x and y positions of the upper left-hand corner arguments to **PIXPOS left** and **PIXPOS right** of the image rectangle defined by **lprcRegion**. If you save the image after a call to this function, it will be saved with the cut.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE (i.e. an NRA mask is active), and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROIdefined by the mask **HIGEAR**. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

For 24-bit images, **lpPixel** must point to 3 bytes where the first byte is red, the second - green and the third - blue. For all other bit depths, **lpPixel** must point to a single byte.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR        hIGear;     /* HIGEAR handle of image  */
AT_ERRCOUNT   nErrcount;  /* # of IG errors on the stack  */
AT_RECT      rcRegion;   /* Rectangular region to cut from image
*/
AT_PIXEL      pixel       /* Pixel value used when filling cut
regions */
pixel = 0;

/* For a currently loaded 1-bit image where black = 0, set the
cut area to black*/
nErrcount = IG_clipboard_cut ( hIGear, &rcRegion, &pixel );
```

IG_clipboard_dimensions

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_dimensions (LPAT_DIMENSION
lpWidth,
   LPAT_DIMENSION lpHeight,
   LPUINT lpBitsPerPixel
   );
```

Arguments:

lpWidth	Far pointer to a variable of type AT_DIMENSION to receive the width in pixels of the image currently in the system clipboard.
lpHeight	Far pointer to a variable of type AT_DIMENSION to receive the height in rows of the image currently on the system clipboard.
lpBitsPerPixel	Far pointer to a variable of type UINT to receive the bit depth, in bits per pixel, of the image currently on the system clipboard.

Description:

This function obtains the dimensions of the image currently in the system clipboard. Prior to calling this function you should call **IG_clipboard_paste_available_ex()**, to verify that there is a pasteable image in the system clipboard.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_DIMENSION   nWidth, nHeight;   /* holds the images's width
and height   */
UINT          nBpp;             /* holds the bits per pixel   */
AT_ERRCOUNT    nErrcount; /* holds athe returned error count */
BOOL           bPasteAvail;     /* TRUE if a pasteable image is
on the clipboard */

 /* If a pasteable image is on the clipboard, get its
dimensions: */
IG_clipboard_paste_available ( &bPasteAvail);
if ( bPasteAvail )
  {
```

```
nErrcount = IG_clipboard_dimensions ( &nWidth, &nHeight, &nBpp
);
  }
```

IG_clipboard_paste

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste (LPHIGEAR lphIGear );
```

Arguments:

lphIGear	Far pointer to a variable of type HIGEAR, to receive the HIGEAR handle of the image created by this operation

Description:

This function creates a **HIGEAR** image by pasting the image on the system clipboard. Prior to calling this function, call **IG_clipboard_paste_available_ex()** to verify that there is a paste-able image in the clipboard.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear; /* Receives the HIGEAR handle the image created
*/
BOOL bPasteAvail;  /* TRUE if a pasteable image is present  */
AT_ERRCOUNT   nErrcount;    /* Holds the returned error count*/

IG_clipboard_paste_available ( &bPasteAvail);
if ( bPasteAvail )
/* Create HIGEAR image from contents of system clipboard:*/
   { nErrcount = IG_clipboard_paste ( &hIGear );
   if ( nErrcount )  { … } /* Process any errors …*/ }
```

IG_clipboard_paste_available_ex

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_available_ex (LPBOOL
lpPasteStatus,LPAT_MODE lpRegionType);
```

Arguments:

lpPasteStatus	A far pointer that returns TRUE if the clipboard contains a region of interest that can be pasted into an image.
lpRegionType	A far pointer that returns the type of region stored in the clipboard. Currently the valid values are `IG_REGION_IS_RECT, IG_REGION_IS_NON_RECT, IG_REGION_IS_NOT_AVAIL`.

Description:

This function tells you whether the clipboard contains a valid image or region of interest (ROI) that can be pasted into an image. It also returns the type of region contained in the clipboard: rectangular, non-rectangular, or not available. If the clipboard does contain a a valid ROI, `lpPasteStatus` returns `TRUE`; if `lpPasteStatus` returns `FALSE`, the region type returned is `IG_REGION_IS_NOT_AVAIL`.

A return value of **`FALSE`** does not necessarily mean that the clipboard is empty. It could mean that the clipboard contains non-valued ROI data or text, or that it contains multimedia data.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

NOTE: To paste a rectangular or non-rectangular ROI into the current image, call **`IG_clipboard_paste_merge_ex()`**.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
BOOL bNRpasteAvail; /* TRUE if a pasteable image is on clipboard
*/
AT_MODE nRegionType; /* type of region on the clipboard */
nErrcount = IG_clipboard_paste_available_ex(&bNRpasteAvail,
&nRegionType);
```

```
if (bNRpasteAvail)
   (…)
```

IG_clipboard_paste_merge_ex

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_merge_ex(HIGEAR  hIGear,
   AT_PIXPOS nLeftPos,
   AT_PIXPOS nTopPos
   );
```

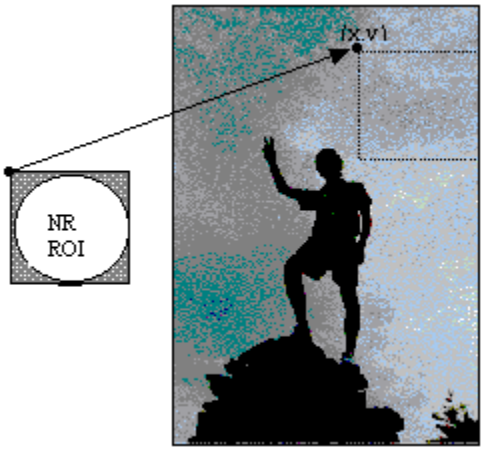
Arguments:

hIGear	Set to the HIGEAR handle of the image in which to merge the clipboard contents.
nLeftPos	X position in HIGEAR image at which to place the upper left corner of the clipboard image when merging.
nTopPos	Y position in HIGEAR image at which to place the upper left corner of the clipboard image when merging.

Description:

This function pastes a rectangular or non-rectangular clipboard image into the **HIGEAR** image that you specify. To check if there is a pasteable image on the clipboard, call **IG_clipboard_paste_available_ex()**.

nLeftPos and **nTopPos** set the (x,y) coordinates of the upper-left corner of the bounding rectangle of the original image. See image below. The image has an (x,y) location within it at which the upper left-corner of the bounding rectangle will be placed. The white circle enclosed within a gray rectangle represents the non-rectangular ROI image stored in the clipboard, where the shaded area represents the transparent area between the bounding rectangle and the non-rectangular ROI. When the clipboard image is merged, only the circle will appear on the image.



NOTE: Use **`IG_clipboard_paste_op_set()`** to tell ImageGear what kind of merge operation to perform.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

NOTE: This function will automatically call **`IG_clipboard_paste_available_ex()`** to confirm that there is data available. Therefore, it is not mandatory to call `IG_clipboard_paste_available_ex()` before making this call, unless you are interested in knowing the type of region contained in the clipboard.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
BOOL      bNRpasteAvail;
AT_PIXPOS xpos, ypos;
AT_MODE   nRegionType;
nErrcount = IG_clipboard_paste_available_ex(&bNRpasteAvail,
&nRegionType);
nErrcount = IG_clipboard_paste_merge_ex(hIGear, xpos, ypos);
```

IG_clipboard_paste_op_get

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_op_get (  HIGEAR hIGear,
   LPAT_MODE lpOperation
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpOperation	A long pointer to an integer constant of type AT_MODE. This will return the current setting for the kind of paste merge operation that will be performed.

Description:

This function returns the current paste-merge operation that will be used when an image from the clipboard is merged into the currently loaded image using **IG_clipboard_paste_merge_ex()**. See the description of **IG_clipboard_paste_op_set()** for the list of possible settings.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image    */
AT_MODE nOperation;  /* current setting for paste-merge
operation */
AT_ERRCOUNT nErrcount    /* # of IG errors currently on the
stack  */
nErrcount = IG_clipboard_paste_op_get (hIGear, &nOperation);
```

IG_clipboard_paste_op_set

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_op_set (HIGEAR hIGear,
   AT_MODE nOperation
   );
```

Arguments:

hIGear	HIGEAR handle of the image that is in the system.
nOperation	An integer constant of type AT_MODE which will be used in future calls to **IG_clipboard_paste_merge_ex()**.

Description:

This function sets the kind of operation to use for future calls to **IG_clipboard_paste_merge_ex()**. **nOperation** is an integer constant of type AT_MODE that is defined in **accucnst.h**. Here are the possible settings and what kind of operation each one will perform on the values of the merging pixels:

IG_ARITH_ADD	Img1 = Img1 + Img2
IG_ARITH_SUB	Img1 = Img1 - Img2
IG_ARITH_MULTI	Img1 = Img1 * Img2
IG_ARITH_DIVIDE	Img1 = Img1 / Img2
IG_ARITH_AND	Img1 = Img1 & Img2
IG_ARITH_OR	Img1 = Img1 | Img2
IG_ARITH_XOR	Img1 = Img1 ^ Img2
IG_ARITH_ADD_SIGN_CENTERED	Img1 = Img1 + SC_Img2
IG_ARITH_NOT	Img1 = ~Img1
IG_ARITH_OVER	Img1 = Img2

You can also set nOperation to 0, which is the default. This will cause the image in the clipboard to just be copied over the currently loaded image – no merging of intersecting pixel values will occur.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR       hIGear;         /* HIGEAR handle of image      */
AT_ERRCOUNT  nErrcount       /* # of IG errors on the stack */
nErrcount = IG_clipboard_paste_op_set( hIGear, IG_ARITH_AND );
```

IG_close

Group: Utility

Declaration:

```
AT_ERRCOUNT IG_close ( LPVOID lpData);
```

Arguments:

lpData	This argument currently isn't used and reserved for the future use. Must be set to **NULL** now.

Description:

When your application no longer needs the use of ImageGear functions, you should close ImageGear using this function. This function frees resources and memory that ImageGear allocated while it was in use. This function should be called before the application program exits. `lpData` must be the same value as one passed to preceding call of **IG_initialize()**.

Bits Per Pixel: Not applicable

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
/* Close ImageGear */
nErrCount = IG_close (lpData);
```

Sample: ImageGearDemo

See also:

IG_initialize() function.

IG_colorspace_conversion_apply_to_image

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_colorspace_conversion_apply_to_image(
HIGCSCONVERTER hCSConverter,
HIGEAR hIGear
    );
```

Arguments:

hCSConverter	Handle of color space converter
hIGear	HIGEAR handle of image to apply conversion to

Description:

This function applies conversion to the entire image. Using this routine is similar to using **IG_image_colorspace_convert()**function, but using the color space conversion engine allows you to manage the bit depths of the result channels.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Use the color space converter to convert an image
   to 24-bit RGB, no alpha or extra channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depths[] = { 8, 8, 8 }; /* Result channel depths */
nErrcount = IG_colorspace_conversion_create_from_image(
    hImage, IG_COLOR_SPACE_ID_RGB, depths, 3, NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_image(
    hCSConv, hImage);
nErrcount = IG_colorspace_conversion_destroy(hCSConv);
```

IG_colorspace_conversion_apply_to_raster

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_colorspace_conversion_apply_to_raster(
HIGCSCONVERTER hCSConverter,
const void* sourceRaster,
void* resultRaster,
AT_INT rasterLength
    );
```

Arguments:

hCSConverter	Handle of color space converter
sourceRaster	Raster to be converted
resultRaster	Raster to store conversion result
rasterLength	Number of PIXELS (not bytes!) in raster to convert

Description:

This function applies conversion to a single raster. `resultRaster` argument should be sufficient to store the converted pixels. If the result raster is run-ends, the store must be suitable to contain the most capacious case. `rasterLength` argument specifies the quantity of pixels (not bytes!) in the raster.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Use the color space converter to convert a CMYK raster
to 24-bit RGB.  Then fill a test image with it and save file. */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depthsCMYK[] = { 8, 8, 8, 8 }; /* CMYK channel depths */
AT_INT depthsRGB[] = { 8, 8, 8 }; /* RGB channel depths */
HIGDIBINFO hDIBCMYK;    /* CMYK DIB info handle */
HIGDIBINFO hDIBRGB;     /* RGB DIB info handle */
AT_DIMENSION x, y, w = 64, h = 64; /* Test image dimensions */
AT_DIMENSION rasterSizeCMYK; /* CMYK raster size */
AT_DIMENSION rasterSizeRGB;  /* RGB raster size */
LPAT_PIXEL lpRasterCMYK;     /* CMYK raster data */
LPAT_PIXEL lpRasterRGB;      /* RGB raster data */

/* Create CMYK test raster (solid yellow) */
nErrcount = IG_DIB_info_create(&hDIBCMYK, w, h,
    IG_COLOR_SPACE_ID_CMYK, 4, depthsCMYK);
rasterSizeCMYK = IG_DIB_info_raster_size_get(hDIBCMYK);
lpRasterCMYK = (LPAT_PIXEL) calloc(rasterSizeCMYK, 1);
for (x = 0; x <= rasterSizeCMYK - 4; x += 4)
    lpRasterCMYK[x] = 255; /* Yellow */

/* Create RGB test image */
nErrcount = IG_DIB_info_create(&hDIBRGB, w, h,
    IG_COLOR_SPACE_ID_RGB, 3, depthsRGB);
rasterSizeRGB = IG_DIB_info_raster_size_get(hDIBRGB);
```

```
lpRasterRGB = (LPAT_PIXEL) malloc(rasterSizeRGB);
nErrcount = IG_image_create(hDIBRGB, &hImage);

/* Convert CMYK test raster to RGB raster */
nErrcount = IG_colorspace_conversion_create_from_dib_info(
    hDIBCMYK, IG_COLOR_SPACE_ID_RGB, depthsRGB, 3,
    NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_raster(
    hCSConv, lpRasterCMYK, lpRasterRGB, w);
nErrcount = IG_colorspace_conversion_destroy(hCSConv);

/* Fill test image with converted raster */
for (y = 0; y < h; y++)
    nErrcount = IG_DIB_raster_set(hImage, y, lpRasterRGB,
        IG_PIXEL_UNPACKED);

/* Save test image to a file and clean up */
nErrcount = IG_save_file(hImage, "test.bmp",
IG_SAVE_BMP_UNCOMP);
nErrcount = IG_DIB_info_delete(hDIBRGB);
nErrcount = IG_DIB_info_delete(hDIBCMYK);
nErrcount = IG_image_delete(hImage);
free(lpRasterCMYK);
free(lpRasterRGB);
```

IG_colorspace_conversion_create

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_colorspace_conversion_create(
enumIGColorSpaceIDs sourceSpace,
const AT_RGBQUAD* sourceSpaceAttribs,
const AT_INT* sourceChannelDepths,
AT_INT sourceChannelCount,
enumIGColorSpaceIDs resultSpace,
const AT_INT* resultChannelDepths,
AT_INT resultChannelCount,
LPCAT_COLORSPACE_CONVERSION_OPTIONS options,
HIGCSCONVERTER* hCSConverter
    );
```

Arguments:

sourceSpace	Color space to convert from
sourceSpaceAttribs	Source color space attributes
sourceChannelDepths	Array of channel depths for source color space
sourceChannelCount	Number of channels in source color space
resultSpace	Color space to convert to
resultChannelDepths	Array of channel depths in resulting color space
resultChannelCount	Number of channels in resulting color space
options	Conversion options
hCSConverter	Handle of color space converter

Description:

This function creates conversion engine based on the source and the result color space attributes. `NULL` is an acceptable value for `resultChannelDepths`. In this case the result channels will be constructed using **`IG_colorspace_conversion_result_channels_build()`** routine. `Options` controls the conversion flow. You may pass `NULL` for `Options` if you don't want to specify any options.

For the detailed information about `AT_COLORSPACE_CONVERSION_OPTIONS` structure and the whole color conversion process please refer to the section Working with Color Space Conversion Engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Use the color space converter to convert an image
   from CMYK to 24-bit RGB, no alpha or extra channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depthsCMYK[] = { 8, 8, 8, 8 }; /* CMYK channel depths */
AT_INT depthsRGB[] = { 8, 8, 8 };     /* RGB channel depths */
nErrcount = IG_colorspace_conversion_create(
    IG_COLOR_SPACE_ID_CMYK, NULL, depthsCMYK, 4,
    IG_COLOR_SPACE_ID_RGB, depthsRGB, 3,
    NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_image(
    hCSConv, hImage);
nErrcount = IG_colorspace_conversion_destroy(hCSConv);
```

IG_colorspace_conversion_create_from_dib_info

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_colorspace_conversion_create_from_dib_info(
HIGDIBINFO hDIB,
enumIGColorSpaceIDs resultSpace,
const AT_INT* resultChannelDepths,
AT_INT resultChannelCount,
LPCAT_COLORSPACE_CONVERSION_OPTIONS options,
HIGCSCONVERTER* hCSConverter
    );
```

Arguments:

hDIB	DIB info handle, describes source color space
resultSpace	Color space to convert to
resultChannelDepths	Array of channel depths in resulting color space
resultChannelCount	Number of channels in resulting color space
options	Conversion options
hCSConverter	Handle of color space converter

Description:

This function creates conversion engine based on source and result color space attributes. `NULL` is an acceptable value for `resultChannelDepths`. In this case the result channels will be constructed using **`IG_colorspace_conversion_result_channels_build()`** routine. The source color space attributes are taken from the specified DIB information handle. `Options` controls the conversion flow. You may pass `NULL` for `Options` if you don't want to specify any options.

For the detailed information about `AT_COLORSPACE_CONVERSION_OPTIONS` structure and the whole color conversion process please refer to the section Working with Color Space Conversion Engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Use the color space converter to convert an image
from CMYK to 24-bit RGB, no alpha or extra channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
HIGDIBINFO hDIB;        /* Source DIB info handle */
AT_INT depths[] = { 8, 8, 8 }; /* Result channel depths */
nErrcount = IG_image_DIB_info_get(hImage, &hDIB);
nErrcount = IG_colorspace_conversion_create_from_dib_info(
    hDIB, IG_COLOR_SPACE_ID_RGB, depths, 3, NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_image(
    hCSConv, hImage);
nErrcount = IG_colorspace_conversion_destroy(hCSConv);
nErrcount = IG_DIB_info_delete(hDIB);
```

IG_colorspace_conversion_create_from_image

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_colorspace_conversion_create_from_image(
HIGEAR                                  hIGear,
enumIGColorSpaceIDs                     resultSpace,
const AT_INT*                           resultChannelDepths,
AT_INT                                  resultChannelCount,
LPCAT_COLORSPACE_CONVERSION_OPTIONS     options,
HIGCSCONVERTER*                         hCSConverter
    );
```

Arguments:

hIGear	HIGEAR handle of image
resultSpace	Color space to convert to
resultChannelDepths	Array of channel depths in resulting color space
resultChannelCount	Number of channels in resulting color space
options	Conversion options
hCSConverter	Handle of color space converter

Description:

This function creates conversion engine based on source and result color space attributes. `NULL` is an acceptable value for `resultChannelDepths`. In this case the result channels will be constructed using

IG_colorspace_conversion_result_channels_build() routine. The color space attributes are taken from specified `hIGear` instance. `Options` controls the conversion flow. You may pass `NULL` for `Options` if you don't want to specify any options.

For the detailed information about `AT_COLORSPACE_CONVERSION_OPTIONS` structure and the whole color conversion process please refer to the section Working with Color Space Conversion Engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Use the color space converter to convert an image
   to 24-bit RGB, no alpha or extra channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depths[] = { 8, 8, 8 }; /* Result channel depths */
nErrcount = IG_colorspace_conversion_create_from_image(
    hImage, IG_COLOR_SPACE_ID_RGB, depths, 3, NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_image(
    hCSConv, hImage);
nErrcount = IG_colorspace_conversion_destroy(hCSConv);
```

IG_colorspace_conversion_destroy

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_colorspace_conversion_destroy(
HIGCSCONVERTER hCSConverter
    );
```

Arguments:

hCSConverter	Handle of color space converter

Description:

This function destroys an instance of the color space conversion engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Use the color space converter to convert an image
   to 24-bit RGB, no alpha or extra channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depths[] = { 8, 8, 8 }; /* Result channel depths */
nErrcount = IG_colorspace_conversion_create_from_image(
    hImage, IG_COLOR_SPACE_ID_RGB, depths, 3, NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_image(
    hCSConv, hImage);
nErrcount = IG_colorspace_conversion_destroy(hCSConv);
```

IG_colorspace_conversion_is_applicable

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_colorspace_conversion_is_applicable(
enumIGColorSpaceIDs                     sourceSpace,
const AT_INT*                           sourceChannelDepths,
AT_INT                                  sourceChannelCount,
enumIGColorSpaceIDs                     resultSpace,
LPCAT_COLORSPACE_CONVERSION_OPTIONS     options,
AT_BOOL*                                applicable
    );
```

Arguments:

sourceSpace	Color space to convert from
sourceChannelDepths	Array of channel depths for source color space
sourceChannelCount	Number of channels in source color space
resultSpace	Color space to convert to
options	Conversion options
applicable	This gets set to TRUE if conversion is possible, otherwise FALSE

Description:

This function checks whether conversion is possible for the given combination of source and result color spaces. In general, conversion cannot be constructed only if: 1) the result color space is indexed and 2) the source color space is something other than indexed or grayscale (8bpp or less). `Options` controls the conversion flow. You may pass `NULL` for `Options` if you don't want to specify any options.

For the detailed information about `AT_COLORSPACE_CONVERSION_OPTIONS` structure and the whole color conversion process please refer to the section Working with Color Space Conversion Engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Can we convert from 48-bit RGB to CMYK? */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
AT_INT depths[] = { 16, 16, 16 }; /* Channel depths */
AT_BOOL bApplicable;    /* Is conversion applicable? */
nErrcount = IG_colorspace_conversion_is_applicable(
    IG_COLOR_SPACE_ID_RGB, depths, 3,
    IG_COLOR_SPACE_ID_CMYK, NULL, &bApplicable);
/* bApplicable is TRUE */
```

IG_colorspace_conversion_result_channels_build

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_colorspace_conversion_result_channels_build(
```

```
enumIGColorSpaceIDs                     sourceSpace,
const AT_INT*                           sourceChannelDepths,
AT_INT                                  sourceChannelCount,
enumIGColorSpaceIDs                     resultSpace,
AT_INT*                                 resultChannelDepths,
AT_INT*                                 resultChannelCount,
LPCAT_COLORSPACE_CONVERSION_OPTIONS     options
    );
```

Arguments:

sourceSpace	Color space to convert from
sourceChannelDepths	Array of channel depths for source color space
sourceChannelCount	Number of channels in source color space
resultSpace	Color space to convert to
resultChannelDepths	Array of channel depths in resulting color space
resultChannelCount	Number of channels in resulting color space
options	Conversion options

Description:

This function builds the channels structure for the specified conversion. It fills `resultChannelDepths` with the calculated depths. `resultChannelCount` contains the quantity of channels in the result raster. `Options` controls the conversion flow. You may pass `NULL` for `Options` if you don't want to specify any options.

For the detailed information about `AT_COLORSPACE_CONVERSION_OPTIONS` structure and the whole color conversion process please refer to the section Working with Color Space Conversion Engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Calculate output channel depths for RGB to CMYK */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
AT_INT depthsIn[] = { 8, 8, 8 }; /* Channel depths */
AT_INT depthsOut[4]; /* Output channel depths */
AT_INT nDepthsOut = 4; /* Number of output channel depths */
nErrcount = IG_colorspace_conversion_result_channels_build(
    IG_COLOR_SPACE_ID_RGB, depthsIn, 3,
    IG_COLOR_SPACE_ID_CMYK, depthsOut, &nDepthsOut, NULL);
/* depthsOut[] is { 8, 8, 8, 8 } */
```

IG_colorspace_conversion_result_palette_get

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_colorspace_conversion_result_palette_get(
HIGCSCONVERTER                          hCSConverter,
AT_RGBQUAD*                             palette,
AT_INT*                                 paletteLength
    );
```

Arguments:

hCSConverter	Handle of color space converter
palette	Array of palette entries
paletteLength	Number of palette entries

Description:

This function builds the result palette. if the result colorspace is indexed, `paletteLength` will be filled with the quantity of palette entries.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Use the color space converter to convert an image
to 8-bit indexed, no alpha or extra channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGCSCONVERTER hCSConv; /* Color space converter handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depths[] = {8};  /* Result channel depths */
AT_RGBQUAD pal[256];    /* Result palette */
AT_INT nEntries;        /* # of palette entries */

nErrcount = IG_colorspace_conversion_create_from_image(
    hImage, IG_COLOR_SPACE_ID_I, depths, 1, NULL, &hCSConv);
nErrcount = IG_colorspace_conversion_apply_to_image(
    hCSConv, hImage);
nErrcount = IG_colorspace_conversion_result_palette_get(
    hCSConv, pal, &nEntries);
```

```
nErrcount = IG_colorspace_conversion_destroy(hCSConv);
```

IG_color_space_level_get

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_color_space_level_get(AT_MODE
nColorSpaceID,
   LPAT_MODE lpnSupportLevel
   );
```

Arguments:

nColorSpaceID	Set this to an AT_MODE constant for the type of color space you would like to get the support level setting of. The names of the appropriate constants begin with IG_COLOR_SPACE_ prefix.
lpnSupportLevel	A far pointer that returns the current setting level for the color space specified in the first argument. See **IG_color_space_level_set()**.

Description:

This function queries the current option level setting for the color space that you specify. You must supply nColorSpaceID with a constant of type AT_MODE from accucnst.h that specifies the color space you wish to query. The second argument will return an ImageGear constant that tells you the current option level setting for the color space.

NOTE: ImageGear fully supports the loading and saving of TIFF-CMYK images. A CMYK image will only be converted to RGB for the purpose of display. CMYK is a color scheme designed for printing and cannot be used for screen display.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

NOTE: For more details see the description for **IG_color_space_level_set()** and the section Working With Color Space Options.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
AT_MODE   nSupportLevel;
nErrcount = IG_color_space_level_get(IG_COLOR_SPACE_CMYK,
&nSupporLevel);
```

IG_color_space_level_set

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_color_space_level_set(AT_MODE
nColorSpaceID, AT_MODE nSupportLevel);
```

Arguments:

nColorSpaceID	Set this to an AT_MODE constant for the type of color space you would like to set the support level setting for. For example: IG_COLOR_SPACE_CMYK.
nSupportLevel	Set this to an AT_MODE constant for level of support for the color space specified in the first argument. For CMYK support, the possible settings are IG_CONVERT_TO_RGB or IG_FULL_SUPPORT.

Description:

This function allows you to set the option level setting for the color space that you specify. You must supply nColorSpaceID with a constant of type **AT_MODE** from accucnst.h that specifies the color space you wish to query, and nSupportLevel with a constant of type **AT_MODE** that specifies the level of support you would like your application to provide.

The CMYK color space is supported using the following settings:

IG_CONVERT_TO_RGB	Loads a CMYK image and converts it to RGB.
IG_FULL_SUPPORT	Full support for loading and saving CMYK images.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

NOTE: ImageGear fully supports the loading and saving of CMYK images. A CMYK image will only be converted to RGB for the purpose of display. CMYK is a color scheme designed for printing and cannot be used for screen display.

NOTE: For more details see the section Working With Color Space OptionsUsing ImageGear.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
nErrcount= IG_color_space_level_set(IG_COLOR_SPACE_CMYK,
IG_COLOR_FULL_SUPPORT);
```

IG_comm_comp_attach

Group: Component Manager

Declaration:

```
AT_ERRCODE  LACCUAPI IG_comm_comp_attach (
    LPCHAR lpCompName
    );
```

Arguments:

lpCompName	The Name of IG Component to be linked with main IG module.

Description:

This function allows you to attach ImageGear component defined by `lpCompName` to the main ImageGear module. By default, ImageGear searches for components in the same directory where main ImageGear module is located but it is possible to specify exact path and filename to given module name of component using global parameters api function **`IG_gctrl_item_set()`** and setting parameter "`COMM.<COMP_NAME>`" to full path of component.

See also the section Using ImageGear Component Manager.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
...
#include "i_ART.h"
...
/*  Initialize ART component   */
IG_comm_comp_attach( "ART" );
...
```

IG_comm_comp_check

Group: Component Manager

Declaration:

```
AT_BOOL    ACCUAPI IG_comm_comp_check(
   LPCHAR lpCompName
   );
```

Arguments:

lpCompName	The Name of IG Component attached to the main IG module.

Description:

This function allows you to check if ImageGear component defined by `lpCompName` argument is currently attached or not. If component is attached it returns `TRUE`, if not - `FALSE`.

See also the section Using ImageGear Component Manager

Bits Per Pixel: N/A

Return Value:

`TRUE` - if component is attached successfully. `FALSE` - if not.

Example:

```
bFoundLZW = IG_comm_comp_check("LZW");

if( bFoundLZW )
{
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_INTERLIVED,
MF_ENABLED|MF_BYCOMMAND );
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_NONINTERLIVED,
MF_ENABLED|MF_BYCOMMAND );

IG_fltr_ctrl_get(IG_FORMAT_GIF, “INTERLACE”, FALSE, NULL, NULL,
(LPVOID)&bInterlaced, sizeof(&bInterlaced));
CheckMenuItem(GetMenu(hWnd), ID_FILE_SAVE_INTERLIVED,
MF_BYCOMMAND | bInterlaced?MF_CHECKED:MF_UNCHECKED);
```

```
CheckMenuItem(GetMenu(hWnd), ID_FILE_SAVE_NONINTERLIVED,
MF_BYCOMMAND | bInterlaced?MF_UNCHECKED:MF_CHECKED);
}else
{
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_INTERLIVED,
MF_GRAYED|MF_BYCOMMAND );
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_NONINTERLIVED,
MF_GRAYED|MF_BYCOMMAND );
}
```

IG_comm_comp_list

Group: Component Manager

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_comm_comp_list(
    LPUINT *lpnCount,
    UINT nIndex,
    LPCHAR lpComp,
    DWORD dwCompSize,
    LPUINT lpnRevMajor,
    LPUINT lpnRevMinor,
    LPUINT lpnRevUpdate,
    LPCHAR lpBuildDate,
    UINT nBDSize,
    LPCHAR lpInfoStr,
    UINT nISSize
    );
```

Arguments:

lpnCount	OUT: The number of attached components
nIndex	IN: The index of component from the list
lpComp	OUT: The buffer where to return the name of component specified by `nIndex` index
dwCompSize	IN: The size of `lpBuffer` in bytes
lpnRevMajor	Far pointer to an `INT` variable in which will be stored the Major version number of the version of the Component specified by `nIndex.`
lpnRevMinor	Far pointer to an `INT` variable in which will be stored the Minor version number of the version of the Component specified by `nIndex.`
lpnRevUpdate	Far pointer to an `INT` variable in which will be stored the Update (bug fix) number, reflecting any updates you have received and installed in this version of the Component specified by `nIndex.`

lpBuildDate	The buffer where to return the build date of the current version of the Component specified by `nIndex`. The return value is a string in the format “Mmm dd yyyy”, such as “Jul 04 2002.”
nBDSize	The size of buffer where `lpBuildDate` is returned.
lpInfoStr	The buffer where to return the info string about the Component specified by `nIndex.`
nISSize	The size of buffer where `lpInfoStr` is returned.

Description:

This function allows to obtain information about currently loaded components. First argument returns actual number of attached components. `nIndex` specifies the index of the component in the components list which name is copied into `lpBuffer`. The rest of parameters return information about component specified by `nIndex`.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`

IG_comm_entry_request

Group: Component Manager

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_comm_entry_request(
    LPCHAR lpEntryName,
    LPAFT_ANY *lpFuncPtr,
    LPCHAR lpReason
    );
```

Arguments:

lpEntryName	The full name of entry "<Comp_name>.<Func_name>", where <Comp_name> is a component name where function is exported and <Func_name> is a name of function.
lpFuncPtr	Pointer where to return pointer to requested function.
lpReason	Optional parameter to specify string description of the reason for this request.

Description:

This function is used to get pointer to the function from component with given name. Usually it is used with component public header i_<COMP_NAME>.h where actual type of function is declared.

See also the section Using ImageGear Component Manager

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS

IG_comm_function_call

Group: Component Manager

Declaration:

```
LONG  CACCUAPI   IG_comm_function_call(
   LPCHAR lpEntryName,
    ...       );
```

Arguments:

lpEntryName	The name of entry in GFT to call
...	additonal parameters

Description:

This function is used to call function from component. Usually it is not used directly but it is used in macro declarations defined in component public headers i_<COMP_NAME>.h.

See also the section Using ImageGear Component Manager

Bits Per Pixel: N/A

Return Value:

Returns a LONG indicating the requested component function.

Example:

See example in "Component Manager API" sectionUsing ImageGear.

IG_convert_DPI_to_PPM

Group: Image Utility

Declaration:

```
LONG ACCUAPI IG_convert_DPI_to_PPM(LONG lDotsPerInch);
```

Arguments:

lDotsPerInch	A variable of type LONG, holding the dpi value that can be converted to pels per meter.

Description:

Converts Dots Per Inch (DPI) to Pels Per Meter (PPM). “Pels” is an abbreviated term for pixels. This function can be useful when you are converting a file that supports dots per inch to a DIB format, which supports pels per meter. The header structure of a DIB (the BIMAPINFOHEADER) contains two fields whose values are defined in “pels per meter”: LONG biXPelsPerMeter and LONG biYPelsPerMeter.

Bits Per Pixel: Not applicable

Return Value:

Returns a LONG indicating the pels per meter of an image.

Example:

```
LONG IPpm, IDpi;
...
IPpm=IG_convert_DPI_to_PPM(IPpi);
...
```

IG_convert_PPM_to_DPI

Group: Image Utility

Declaration:

```
LONG ACCUAPI IG_convert_PPM_to_DPI(LONG lPelsPerMeter);
```

Arguments:

lPelsPerMeter	A variable of type LONG, holding the pels per meter value that can be converted to dots per inch.

Description:

Converts Pels Per Meter (PPM) to Dots Per Inch (DPI). “Pels” is an abbreviated term for pixels. This function can be useful when you are converting a DIB to a format that supports dots per inch. The header structure of a DIB (the `BIMAPINFOHEADER`) contains two fields whose values are defined in “pels per meter”: LONG `biXPelsPerMeter` and LONG `biYPelsPerMeter`.

Bits Per Pixel: Not applicable

Return Value:

Returns a LONG indicating the DPI of an image.

Example:

```
LONG IPpm, IDpi;
...
IPpm=IG_convert_PPM_to_DPI(IPpm);
...
```

IG_cpm_image_embedded_profile_check

Group: Color Profile

Declaration:

```
AT_ERRCODE ACCUAPI IG_cpm_image_embedded_profile_check(HIGEAR
hIGear, LPAT_BOOL lpbEmbedded);
```

Arguments:

hIGear	(in) A handle of the image to check.
lpbEmbedded	(out) Return TRUE if profile embedded, otherwise, FALSE.

Description:

This function checks to see whether the image has embedded color profile.

Bits Per Pixel:

All

Return Value:

Return value is a code of last error or NULL if success.

Example:

```
HIGEAR hIGear;/* Handle of the image to check*/
AT_BOOL bEmbeddedProfile = FALSE;/* Flag to return whether
profile embedded or not */

/* Load an image into hIGear */
......

/* Check whether the image has embedded profile. */
if (IGE_SUCCESS == IG_cpm_image_embedded_profile_check(hIGear,
&bEmbeddedProfile))
{
……
}
```

Sample:

GUI

IG_cpm_image_profile_get

Group: Color Profile

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_image_profile_get(
   HIGEAR hIGear,
   LPAT_BOOL lpbIsLocal,
   LPAT_MODE lpnColorSpace,
   LPCHAR lpStatusStr,
   UINT nStatusSize
   LPUINT lpnStatusLen,
   LPDWORD lpnProfileSize
   LPBYTE lpProfileData,
   DWORD dwProfileDataSize
   );
```

Arguments:

hIGear	IN: handle of image where to set profile.
lpbIsLocal	OUT: return TRUE if local profile associated with given image and FALSE in other case. If NULL then argument ignored.
lpnColorSpace	OUT: color space id of returned profile. Possible returned values: IG_COLOR_SPACE_RGB IG_COLOR_SPACE_CMYK If NULL then parameter ignored.
lpStatusStr	OUT: pointer where to copy textual information about local profile. If NULL then parameter ignored.
nStatusSize	IN: size of lpStatusStr buffer.
lpnStatusLen	OUT: Return length of actual status message. If NULL then parameter ignored.
lpnProfileSize	OUT: If not NULL then returns actual profile size in ICC format.
lpProfileData	OUT: Pointer where to put profile data. Profile will be written according to ICC.1:1998-09 specification.
dwProfileDataSize	IN: the size of lpProfileData buffer.

Description:

This function returns information about color profile associated with given image. Second parameter is used to get information either this image has embedded local profile or use global profile.

The rest of parameters are used to provide text description of profile and profile data iteslf. The profile returned through lpProfileData will be written according to ICC specification.

Depending from parameters it either return information about profile or profile data itself. For example, if lpnProfileSize is not NULL then the size of profile is calculated and returned. If lpProfileData is not NULL then profile is encoded into standard ICC format and returned through this parameter.

Please note that color profile management is disabled by default. See section Working with ImageGear Color Profile Manager for descirption how to activate it.

Bits Per Pixel: ALL

Return Value:

Return value is a code of last error or NULL if success.

IG_cpm_image_profile_set

Group: Color Profile

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_image_profile_set(
    HIGEAR hIGear,
    LPAT_BYTE lpRawData,
    DWORD dwRawSize,
    AT_BOOL bConvert
    );
```

Arguments:

hIGear	IN: handle of image where to set profile.
lpRawData	IN: raw data of new profile. Can be either NULL or pointer to memory buffer that contains valid color profile in format specified by ICC.1:1998-09.
dwRawSize	IN: lenth of data stored in `lpRawData`.
bConvert	IN: if TRUE then convert all images associated with old profile to new profile, but if FALSE then simple replace profile without any conversion.

Description:

This function provides color profile management for given particular image. There can be two ways how an image is associated with color profile:

- First way: image itself does not store color profile data but use profile from global parameters (WCP).
- Second way: image stores color profile locally and does not depend on global settings.

If `lpRawData` is `NULL` during this function call, then previous local profile (if it existed) is deleted and image become dependent on global profile correspondent to color space used by its pixel data. If `lpRawData` is valid ICC profile then previous local profile (if it existed) is deleted and image becomes associated with new local profile.

The last parameter specifies how pixel data should be changed during profile change operation. If `bConvert` is `TRUE`, then pixel is converted from one format to another but in other case pixel data is unchanged.

Please note, that color profile management is disabled by default. See section Working with ImageGear Color Profile Manager for descirption how to activate it.

If some error happens such as invalid or unsupported profile or color space mismatch in image and color profile, then function returns appropriate error code.

Bits Per Pixel: ALL

Return Value:

Return value is a code of last error or `NULL` if success.

IG_cpm_profile_get

Group: Color Profile

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_profile_get(
   AT_MODE nColorSpace,
   DWORD nProfileGroup,
   LPCHAR lpStatusStr,
   UINT nStatusSize
   LPUINT lpnStatusLen,
   LPDWORD lpnProfileSize,
   LPBYTE lpProfileData,
   DWORD dwProfileDataSize
   );
```

Arguments:

nColorSpace	IN: color space ID which profile status to get. Possible value:
	`IG_COLOR_SPACE_RGB`
	`IG_COLOR_SPACE_CMYK`

nProfileGroup	IN: color profile group where profiles to get. Possible value:
	`IG_CP_GRP_DISPLAY`
	`IG_CP_GRP_WORKING`
	`IG_CP_GRP_EXPORT`
	`IG_CP_GRP_IMPORT`

lpStatusStr	OUT: pointer where to copy textual information about local profile. If `NULL` then parameter is ignored.
nStatusSize	IN: size of `lpStatusStr` buffer.
lpnStatusLen	OUT: return length of actual status message. If `NULL` then parameter ignored.
lpnProfileSize	OUT: If this parameter is not `NULL` then actual size of color profile in ICC format is returned.
lpProfileData	OUT: Pointer where to put profile data or ignored if parameter is `NULL`. Profile will be written according to ICC.1:1998-09 specification.
dwProfileDataSize	IN: the size of `lpProfileData`

Description:

This function returns information about global profile for given color space of given group. Depending from parameters it either returns information about profile or profile data itself. For example, if `lpnProfileSize` is not `NULL`, then size of profile is calculated and returned. If `lpProfileData` is not `NULL`, then profile is encoded into standard ICC format and returned through this parameter.

Please note that color profile management is disabled by default. See section Working with ImageGear Color Profile Manager for descirption how to activate it.

Bits Per Pixel: ALL

Return Value:

Return value is a code of last error or `0` if success.

IG_cpm_profile_set

Group: Color Profile

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_profile_set(
   AT_MODE nColorSpace,
```

```
DWORD nProfileGroup
LPAT_BYTE lpRawData,
DWORD dwRawSize,
AT_BOOL bConvert
);
```

Arguments:

nColorSpace	IN: color space ID which profile to replace. Current supported values:
	`IG_COLOR_SPACE_RGB`
	`IG_COLOR_SPACE_CMYK`

nProfileGroup	IN: color profile group where to set profile. Possible values:
	`IG_CP_GRP_WORKING`
	`IG_CP_GRP_IMPORT`
	`IG_CP_GRP_EXPORT`
	`IG_CP_GRP_DISPLAY`

lpRawData	IN: raw data of new profile. Can be either `NULL` or pointer to memory buffer that contains valid color profile in format specified by ICC.1:1998-09.
dwRawSize	IN: lenth of data stored in `lpRawData`.
bConvert	IN: if TRUE then this function converts all images associated with old profile to new profile, but if FALSE it simply replaces profile without any conversion

Description:

This function sets new value of color profile associated with color space given by `nColorSpace` parameter to profile group given by `nProfileGroup` parameter. If last parameter is set to `TRUE` then all associated images with previous format are converted to new color profile. In the case if `nColorSpace` does not match the same color space that is used in color profile or color profile is in invalid or unsupported format, then error is returned.

Please note that color profile management is disabled by default. See section Working with ImageGear Color Profile Manager for descirption how to activate it.

Bits Per Pixel: All

Return Value:

Return value is a code of last error or `0` if success.

IG_cpm_profiles_reset

Group: Color Profile

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_profiles_reset(
    AT_BOOL bConvert
    );
```

Arguments:

bConvert	IN: specify how this operation should affect associated image.

Description:

This function resets all default color profiles to default values taken from global parameters. If value of parameter is TRUE then it converts all images associated with all global profiles to default profiles, but if FALSE then all global profiles are reset to default values but images are not changed.

Please note that color profile management is disabled by default. See section Working with ImageGear Color Profile Manager for descirption how to activate it.

Bits Per Pixel: ALL

Return Value:

Return value is a code of last error or `0` if success.

IG_DIB_area_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_area_get (HIGEAR hIGear,
    const LPAT_RECT lpRect,
    LPAT_PIXEL lpPixel,
    AT_MODE nPixelFormat
    );
```

Arguments:

hIGear	hIGear handle of an image.
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image bitmap to get
lpPixel	Far pointer to first in an array of bytes large enough to receive all pixels in the area
nPixelFormat	A constant such as IG_DIB_AREA_UNPACKED, specifying in what form you want the pixels stored in your array. The list of IG_DIB_AREA_ constants available is in file accucnst.h.

Description:

This function obtains all the pixels contained within the rectangular portion of image **hIGear** specified by **lpRect**. Use argument **lpPixel** to tell ImageGear where to store the pixels.

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

Use nPixelFormat = IG_DIB_AREA_DIB if you want the data in standard uncompressed DIB format, and with each row returned to you padded to a multiple of 4 bytes length. 1-bit pixels are returned 8 to the byte, most significant bit first. 4-bit pixels are returned 2 to the byte, similarly left justified. 24-bit pixels are returned 3 bytes per pixel, ordered Blue-Green-Red.

Use nPixelFormat = IG_DIB_AREA_UNPACKED if you want the pixels returned 1 per byte (but still 3 bytes for a 24-bit pixel, ordered Blue-Green-Red). Each 1-bit or 4-bit pixel will be returned right justified in a single byte, padded with zeroes in the most significant bits of the byte.

In either case, be sure your area pointed to by lpPixel is large enough to receive all the pixel data including padding.

NOTE: See also function **IG_DIB_area_set()**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR          hIGear;       /* HIGEAR handle of image  */
```

```
AT_RECT        rcBlock;      /* The rectangular block to get  */
AT_PIXEL       cPixArray[400];/* Will receive returned pixels
*/
AT_DIMENSION   nWid, nHi;     /* Will receive width, height of
image     */
UINT           nBpp;          /* Bits per pixel  */
AT_ERRCOUNT  nErrcount;  /* Will receive returned error counts
*/
/* Will fetch upper left 20 x 20 pixels, to cPixArray[]:   */
rcBlock.top = rcBlock.left = 0;
rcBlock.bottom = rcBlock.right = 20;  /* 20x20 area, 400 pixels
*/
nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp
);  */
if ( nErrcount == 0 )      /* If valid image, dimensions
obtained'    */
   {
   if ( (nBpp <= 8) && (nWid >= 20) && (nHi >= 20) )
      {/* (Array is too small for 24-bit)*/
      nErrcount = IG_DIB_area_get ( hIGear, &rcBlock,
         &cPixelArray[0], IG_DIB_AREA_UNPACKED );
      }
   }
```

IG_DIB_area_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_area_set (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const LPAT_PIXEL lpPixel,
   AT_MODE nPixelFormat
   );
```

Arguments:

hIGear	ImageGear `HIGEAR` handle of image
lpRect	Far pointer to an `AT_RECT` struct specifying the rectangular portion of the image bitmap to set
lpPixel	Far pointer to first byte of your pixel data
nPixelFormat	A constant such as `IG_DIB_AREA_UNPACKED` specifying in what form you are providing the pixels. The `IG_DIB_AREA_...` constants are listed in file `accucnst.h`.

Description:

This function transfers pixels from the location pointed to by `lpPixel` into the rectangular portion of the image specified by rectangle `lpRect`.

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

- Use `nPixelFormat = IG_DIB_AREA_DIB` if you are providing the pixels in standard uncompressed DIB format. This means 1-bit or 4-bit pixels will be packed 8 to the byte or 2 to the byte respectively, left justified (first pixel uses most significant bit). 24-bit pixels are in 3 bytes, ordered Blue-Green-Red.
- Use `nPixelFormat = IG_DIB_AREA_UNPACKED` if you are providing the pixels 1 pixel per byte (however, 3 bytes for a 24-bit pixel, ordered Blue-Green-Red). In this case, you provide 1-bit and 4-bit pixels one to a byte, right justified in the byte.

NOTE: See also function **`IG_DIB_area_get()`**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HIGEAR          hIGear;        /* HIGEAR handle of image */
AT_RECT         rcBlock;      /* The rectangular block to set
*/
AT_PIXEL        cPixelArray[400];    /* The pixels to set */
AT_DIMENSION    nWid, nHi;     /* Receives the width & height of
an image */
```

```
UINT            nBpp;          /* Bits per pixel   */
AT_ERRCOUNT     nErrcount;      /* Receives the returned error
counts  */
/* Sets the upper left 20 x 20 pixels, to cPixArray[]:
*/
rcBlock.top = rcBlock.left = 0;
rcBlock.bottom = rcBlock.right = 20;  /* 20x20 area, 400 pixels
*/
nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp
);  */
if ( nErrcount == 0 )      /* If valid image, dimensions
obtained:  */
   {
   if ( (nBpp <= 8) && (nWid >= 20) && (nHi >= 20) )
{
/*Array is too small for 24-bit)     */
   INT           row, col; pix;     /* For the loops below  */
   AT_PIXEL      nPixval;          /* pixel value to set  */
   if (nBpp == 8) nPixval = 128;   /* Value to set if 8-bit   */
   if (nBpp == 4) nPixval = 8;     /* Value to set if 4-bit   */
   if (nBpp == 1) nPixval = 1;     /* Pixel ON if 1-bit    */
      for ( pix=0,row=0; row<20; row++ ) /* For all pixels in
*/
         for ( col=0; col<20; col++ )/* the 20 x 20 array:  */
            cPixelArray[pix++] = nPixval;
            /* Set unpacked in byte    */
      nErrcount = IG_DIB_area_set ( hIGear, &rcBlock,
&cPixelArray[0], IG_DIB_AREA_UNPACKED );
}
   }
```

IG_DIB_area_size_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_area_size_get(HIGEAR hIGear,
   LPAT_RECT lpRect,
   AT_MODE nFormat,
   LPAT_DIMENSION lpSize
   );
```

Arguments:

hIGear	HIGEAR handle of image.
lpRect	A far pointer to a rectangular array of pixels from the image. Setting this to NULL selects the whole image.
nFormat	A variable of type AT_MODE, such as IG_DIB_AREA_DIB, that defines how the data should be stored: packed or unpacked. (They are defined in accucnst.h)
lpSize	A far pointer to a variable that returns the size in bytes of the array of pixels.

Description:

This function calculates and returns the number of bytes required to hold a rectangular region selected from an image. You can use this function to determine the size of the block of memory to allocate to hold the pixel values from the rectangular region. This will help you to avoid data overflow. The value returned by lpSize includes the allocation of space for raster buffering at the end of each raster line. (See the section Device-Independent Bitmaps (DIB) Understanding Bitmap Images.)

Use the format parameter to indicate the form in which you want to store the pixels:

Use:	To:
IG_DIB_AREA_DIB	pad rows to long boundaries (the way they are stored in a DIB).
IG_DIB_AREA_UNPACKED	store pixels 1 per byte for 1, 4, 8-bit images, 1 per 3 bytes for a 24-bit image

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

IG_DIB_bit_depth_get

Group: DIB Info Services

Declaration:

```
AT_INT ACCUAPI IG_DIB_bit_depth_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the bit depth of an image. This number is the sum of the channel bit depths.

For example, if the image is a simple 24-bit RGB image, this function will return 24. If it's a 24-bit RGB image with an 8-bit alpha channel, this function will return 32.

Bits Per Pixel: All

Return Value:

Returns image bit depth.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT imageDepth;      /* Returned bit depth of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
imageDepth = IG_DIB_bit_depth_get(hDIBInfo);
```

IG_DIB_channel_count_get

Group: DIB Info Services

Declaration:

```
AT_INT ACCUAPI IG_DIB_channel_count_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the number of channels in the image.

For example, a typical 24-bit RGB image has three 8-bit channels (red, green, blue) so this function will return 3. For a 24-bit RGB image with a single alpha channel, this function would return 4.

Bits Per Pixel: All

Return Value:

Returns the number of channels in the image.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT nChannels;       /* Returned number of channels */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nChannels = IG_DIB_channel_count_get(hDIBInfo);
```

IG_DIB_channel_depth_get

Group: DIB Info Services

Declaration:

```
AT_INT ACCUAPI IG_DIB_channel_depth_get(
   HIGDIBINFO hDIB,
   AT_INT Index
   );
```

Arguments:

hDIB	DIB info handle
Index	Index of channel to return bit depth of

Description:

This function returns the bit depth of the channel specified by the `Index` parameter.

For example, if you had a typical 24-bit RGB image with three 8-bit channels, you could specify 0, 1, or 2 for the index and the return value would be 8.

Bits Per Pixel: All

Return Value:

Returns specified channel's bit depth.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depth;           /* Returned depth of first channel */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
depth = IG_DIB_channel_depth_get(hDIBInfo, 0);
```

IG_DIB_channel_depths_get

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_channel_depths_get(
   HIGDIBINFO hDIB,
   AT_INT ChannelCount,
   AT_INT *lpChannelDepths
   );
```

Arguments:

hDIB	DIB info handle
ChannelCount	Number of channels to return depths for
lpChannelDepths	Array to copy channel depths to

Description:

This function copies channel bit depths to an array. You can use it to find out the bit depths of each individual channel.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT nChannels;       /* Number of channels in image */
LPAT_INT depths;        /* Array of channel depths */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nChannels = IG_DIB_channel_count_get(hDIBInfo);
depths = (LPAT_INT) malloc(nChannels * sizeof(AT_INT));
nErrcount = IG_DIB_channel_depths_get(hDIBInfo, nChannels,
depths);
```

IG_DIB_colorspace_get

Group: DIB Info Services

Declaration:

```
enumIGColorSpaceIDs ACCUAPI IG_DIB_colorspace_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the image's color space. This is a bitmask that indicates the color space as well as the presence of alpha, premultiplied alpha, and extra channels.

See the IG_util_colorspace_...() functions for more functions that retrieve information about color spaces, alpha, and extra channels.

NOTE: Use caution when extracting information from this bitmask. For example, if you want to look at only the color space, you must use the mask `IG_COLOR_SPACE_ID_ColorMask` as in the example below.

Bits Per Pixel: All

Return Value:

Returns a combination of values from `enumIGColorSpaceIDs`.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
```

```
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
enumIGColorSpaceIDs colorspace; /* Color space of image */
AT_BOOL bIndexed;       /* Is color space of image indexed? */
AT_BOOL bHasAlpha;      /* Does image have an alpha channel? */
/* Find out if an image is indexed, and if it has alpha */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
colorspace = IG_DIB_colorspace_get(hDIBInfo);
bIndexed = (colorspace & IG_COLOR_SPACE_ID_ColorMask) ==
IG_COLOR_SPACE_ID_I;
bHasAlpha = colorspace & IG_COLOR_SPACE_ID_A;
```

IG_DIB_column_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_column_get ( HIGEAR hIGear,
   AT_PIXPOS nX,
   AT_PIXPOS nY1,
   AT_PIXPOS nY2,
   LPAT_PIXEL lpPixel,
   AT_DIMENSION nLenBytes,
   LPAT_DIMENSION lpNumPixels
   );
```

Arguments:

hIGear	`HIGEAR` handle of image
nX	X offset (X coord) of the vertical pixel column to get
nY1	Raster line number at which the vertical column starts
nY2	Raster line number at which the vertical column ends
lpPixel	Far pointer to your data area to which the pixels should be returned
nLenBytes	Size of the data area, in bytes
lpNumPixels	Far pointer to a `AT_DIMENSION` variable in which will be returned the number of pixels (not bytes) transferred

Description:

This function obtains a *column* of pixels from the DIB image bitmap of image **`hIGear`**, and stores the pixels in your data area pointed to by **`lpPixel`**. The offset of the column to retrieve is indicated by **`nX`**. The beginning and ending raster lines of the column are **`nY1`** and **`nY2`**, respectively (top to bottom).

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the pixels will not fit in **nLenBytes**, only **nLenBytes** of data will be transferred. The actual number of pixels transferred is returned in the variable pointed to by **lpNumPixels**.

If the image is 1-bit or 4-bit, the pixels will be returned one to a byte, right justified in the byte. If the image is 24-bit, each pixel will occupy 3 bytes, in Blue-Green-Red order.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HIGEAR  hIGear;      /* HIGEAR handle of image*/
AT_PIXEL cPixArray[400];/* Receives the returned pixels*/
AT_PIXPOS nCol, nYtop, nYbot; /* Column and which rows to get*/
AT_DIMENSION nFetched;/* Holds count of pixels retrieved*/
AT_ERRCOUNT nErrcount;/* Receives returned error counts*/
nCol  = 0;           /* Fetch left boundary of image*/
nYtop = 10;  nYbot = 59;/* 50 pixels, from lines 10 thru 59*/
nErrcount = IG_DIB_column_get ( hIGear, nCol, nYtop, nYbot,
&cPixArray[0], 400, &nFetched );
```

IG_DIB_column_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_column_set ( HIGEAR hIGear,
   AT_PIXPOS nX,
   AT_PIXPOS nY1,
   AT_PIXPOS nY2,
   const LPAT_PIXEL lpPixel,
   AT_DIMENSION nNumPixels
   );
```

Arguments:

hIGear	HIGEAR handle of image
nX	X offset (X coord) of the vertical pixel column to be set
nY1	Raster line number at which the vertical column starts
nY2	Raster line number at which the column ends
lpPixel	Far pointer to first byte of your pixel data
nNumPixels	Number of pixels (not bytes) to transfer

Description:

This function sets the pixel data you supply into the DIB image bitmap *column* specified by **nX**. The pixel at **nX** is set in pixel rows **nY1** through **nY2** inclusive. **nNumPixels** is the number of pixels to set, and should equal (**nY2-nY1+1**).

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image is 1-bit or 4-bit, your pixels should be one to a byte and right justified, beginning at **lpPixel**. If the image is 24-bit, each pixel should occupy 3 bytes, in Blue-Green-Red order.

NOTE: If the image you are modifying is 1-bit, you will probably need to convert the image from run-end encoded to a standard DIB before you can set pixel values. Please see the section Using Pixel Access Functions with ImageGear Using ImageGear for details.

Bits Per Pixel:

1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of image           */
AT_PIXEL cPixArray[400];/* Receives the returned pixels     */
AT_PIXPOS nCol,  nYtop,  nYbot; /* Column and rows to get
*/
AT_DIMENSION nFetched;/* Holds the count of pixels retrieved*/
AT_ERRCOUNT nErrcount;/* Receives the returned error counts */
```

```
/* Restore the pixels saved by the call in example
IG_DIB_column_get:   */
nCol  = 0;          /* Restores to image's left boundary */
nYtop = 10;  nYbot = 59;/* 50 pixels, from lines 10 thru 59*/
nFetched = nYbot - Top + 1;
nErrcount = IG_DIB_column_set ( hIGear, nCol, nYtop, nYbot,
&cPixArray[0],nFetched );
```

IG_DIB_height_get

Group: DIB Info Services

Declaration:

```
AT_DIMENSION ACCUAPI IG_DIB_height_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the height of the image.

Bits Per Pixel: All

Return Value:

Returns image height.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_DIMENSION height;    /* Returned height of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
height = IG_DIB_height_get(hDIBInfo);
```

IG_DIB_info_copy

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_info_copy(
```

```
    HIGDIBINFO hDIBSrc,
    HIGDIBINFO *lphDIBDst
    );
```

Arguments:

hDIBSrc	DIB info handle to copy from
lphDIBDst	Pointer to where copied DIB info handle will be stored

Description:

This function makes a copy of a DIB info object.

NOTE: It does not copy pixel data.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGDIBINFO hDIBInfo;     /* Handle of DIB info to be copied */
HIGDIBINFO hDIBInfoCopy; /* Handle of DIB info copy */
HIGEAR hImage;           /* HIGEAR handle of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nErrcount = IG_DIB_info_copy(hDIBInfo, &hDIBInfoCopy);
```

IG_DIB_info_create

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_info_create(
    HIGDIBINFO *lphDIB,
    AT_DIMENSION width,
    AT_DIMENSION height,
    enumIGColorSpaceIDs colorspace,
    AT_INT channelCount,
    AT_INT *channelDepths
    );
```

Arguments:

lphDIB	Pointer to where created DIB info's handle will be stored
width	Width of image in pixels
height	Height of image in pixels
colorspace	Colorspace of image
channelCount	Number of channels in image
channelDepths	Array of channel depths

Description:

This function creates a new DIB info object and initializes it to the given values. The DIB info object must be deleted with IG_DIB_info_delete when it is finished being used.

NOTE: It does not allocate pixel data storage.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Create a DIB info object describing a 48-bit RGB image */
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGDIBINFO hDIBInfo;             /* DIB info handle */
AT_INT depths[3] = {16, 16, 16}; /* Array of channel depths */
nErrcount = IG_DIB_info_create(&hDIBInfo, 320, 240,
IG_COLOR_SPACE_ID_RGB, 3, depths);
/* ... */
nErrcount = IG_DIB_info_delete(hDIBInfo);
```

IG_DIB_info_delete

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_info_delete(
HIGDIBINFO hDIB
);
```

Arguments:

hDIB	DIB info handle

Description:

This function deletes a DIB info object.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Create a DIB info object describing a 24-bit RGB image */
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGDIBINFO hDIBInfo;             /* DIB info handle */
AT_INT depths[3] = {8, 8, 8};    /* Array of channel depths */
nErrcount = IG_DIB_info_create(&hDIBInfo, 320, 240,
IG_COLOR_SPACE_ID_RGB, 3, depths);
/* ... */
nErrcount = IG_DIB_info_delete(hDIBInfo);
```

IG_DIB_info_raster_size_get

Group: DIB Info Services

Declaration:

```
AT_INT ACCUAPI IG_DIB_info_raster_size_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the number of bytes per raster in the DIB.

Bits Per Pixel: All

Return Value:

Returns DIB raster size.

Example:

```
HIGDIBINFO hDIBInfo;    /* DIB info handle */
AT_INT rasterSize;      /* Returned raster size */
rasterSize = IG_DIB_info_raster_size_get(hDIBInfo);
```

IG_DIB_legacy_bit_depth_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_legacy_bit_depth_get(
HIGEAR hIGear,
LPAT_INT lpBitsPerPixel
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpBitsPerPixel	Returned legacy bit depth of the image

Description:

This function returns the bit depth that earlier ImageGear versions used to store this image. This can be used for working with pixel access functions in the legacy mode.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
AT_INT bpp;             /* Image bit depth */
nErrcount = IG_DIB_legacy_bit_depth_get(hImage, &bpp);
/* bpp could be 1, 4, 8, 9-16, 24, or 32 */
```

IG_DIB_line_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_line_get ( HIGEAR hIGear,
    PIXPOS nX1,
    AT_PIXPOS nY1,
    AT_PIXPOS nX2,
    AT_PIXPOS nY2,
    LPAT_PIXEL lpPixel,
    const AT_DIMENSION nlenOfArray,
    LPAT_DIMENSION lpNumPixels
    );
```

Arguments:

hIGear	`HIGEAR` handle of image
nX1	X coordinate of the first endpoint of line to get
nY1	Y coordinate of the first endpoint
nX2	X coordinate of the second endpoint of line
nY2	Y coordinate of the second endpoint
lpPixel	Far pointer to your data area to which the pixels should be returned
nLenOfArray	Length, in bytes, of `lpPixel` block
lpNumPixels	Far pointer to an `AT_DIMENSION` variable in which the number of pixels (not bytes) transferred is returned.

Description:

This function obtains an arbitrary line of pixels from the DIB image bitmap of the image referenced by `hIGear`. The line does not have to be horizontal or vertical. **`nX1`**,**`nY1`** are the coordinates of one endpoint of the line and `nX2`,`nY2` are of the other.

If the pixel data would overflow your area (`nLenOfArray`), the transfer will be truncated; your data area will not be overflowed. The actual number of pixels returned will be stored in your variable pointed to by `lpNumPixels`.

Bits Per Pixel:

1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during the function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

NOTE: To determine the number of pixels that your line will be comprised of, use the following formula:
max((1+abs(x2-x1)), (1+abs(y2-y1)))

1-bit and 4-bit pixels are returned one to a byte, right justified. 24-bit pixels are returned 3 bytes per pixel, in Blue-Green-Red order.

Example:

See the example for function **IG_DIB_line_set()**.

IG_DIB_line_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_line_set ( HIGEAR hIGear,
   PIXPOS nX1,
   AT_PIXPOS nY1,
   AT_PIXPOS nX2,
   AT_PIXPOS nY2,
   const LPAT_PIXEL lpPixel,
   AT_DIMENSION nNumPixels
   );
```

Arguments:

hIGear	HIGEAR handle of image to transfer pixels to
nX1	X coordinate of the first endpoint of line to store
nY1	Y coordinate of the first endpoint
nX2	X coordinate of the second endpoint of line to store
nY2	Y coordinate of the second endpoint
lpPixel	Far pointer to your data area containing the pixels to be stored
nNumPixels	Number of pixels (not bytes) to be transferred

Description:

This function stores an arbitrary line of pixels in the DIB image bitmap of the image referenced by **HIGEAR**. The line does not have to be horizontal or vertical. (**nX1**,**nY1**) are the coordinates of one endpoint of the line and (**nX2**,**nY2**) are of the other.

NOTE: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image you are modifying is 1-bit, you must convert the image from run-end encoded to a standard DIB before you can set pixel values. Please see the section Using Pixel Access Functions with ImageGear for details.

1-bit and 4-bit pixels should be provided one to a byte, right justified (that is, in the least significant bits of the byte). 24 bit pixels should be 3 bytes per pixel, in Blue-Green-Red order.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

TIP: To calculate the number of pixels that your line will consist of, use the following formula:
max((1+abs(x2-x1)), (1+abs(y2-y1)))

Example:

```
HIGEAR  hIGear;      /* HIGEAR handle of image             */
AT_PIXEL cPixArray[400]; /* Receives the returned pixels
*/
```

```
AT_PIXPOS nXleft, nYtop,  /* Coordinates of upper-left end of
line*/
       nXright, nYbot; /* Coordinates of the lower-right end
*/
AT_DIMENSION nFetched;   /* Holds the count of pixels retrieved
*/
AT_ERRCOUNT nErrcount;   /* Receives the returned error counts
*/
/* Fetch a diagonal line, from Coordinates (0,0) to (100,100):
*/
nXleft  = 0;    nYtop = 0;/* Diagonal line from upper left
corner     */
nXright = 100;  nYbot = 100;
nErrcount = IG_DIB_line_get ( hIGear, nXleft, nYtop, nXright,
nYbot, &cPixArray[0], 400, &nFetched );
 …
/* Now restore the line: */
nErrcount = IG_DIB_line_set ( hIGear, nXleft, nYtop, nXright,
nYbot, &cPixArray[0], nFetched );
```

IG_DIB_palette_alloc

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_palette_alloc(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function allocates a palette for the given DIB.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT nEntries;        /* Number of palette entries */
LPAT_RGBQUAD lpPalette; /* Pointer to palette data */
```

```
AT_INT i;               /* Index for palette loop */
/* Make a palette in which every color is GREEN */
nErrcount = IG_DIB_palette_alloc(hDIB);
nEntries = IG_DIB_palette_length_get(hDIB);
lpPalette = IG_DIB_palette_pointer_get(hDIB);
for (i = 0; i < nEntries; i++)
{
    lpPalette[i].rgbRed = lpPalette[i].rgbBlue = 0;
    lpPalette[i].rgbGreen = 255;
}
```

Sample: None

IG_DIB_palette_length_get

Group: DIB Info Services

Declaration:

```
AT_INT ACCUAPI IG_DIB_palette_length_get(
    HIGDIBINFO hDIB
    );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the number of entries in the DIB's palette.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level

Return Value:

Returns the number of palette entries.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT nEntries;        /* Number of palette entries */
LPAT_RGBQUAD lpPalette; /* Pointer to palette data */
AT_INT i;               /* Index for palette loop */
/* Make a palette in which every color is GREEN */
nErrcount = IG_DIB_palette_alloc(hDIB);
nEntries = IG_DIB_palette_length_get(hDIB);
```

```
lpPalette = IG_DIB_palette_pointer_get(hDIB);
for (i = 0; i < nEntries; i++)
{
    lpPalette[i].rgbRed = lpPalette[i].rgbBlue = 0;
    lpPalette[i].rgbGreen = 255;
}
```

IG_DIB_palette_pointer_get

Group: DIB Info Services

Declaration:

```
LPAT_RGBQUAD ACCUAPI IG_DIB_palette_pointer_get(
    HIGDIBINFO hDIB
    );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns a pointer to the DIB's palette, if it is present. Otherwise it returns `NULL`. You can use this to get and set palette entries or the palette as a whole.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level

Return Value:

Pointer to the DIB palette, if it is present. NULL - otherwise.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT nEntries;        /* Number of palette entries */
LPAT_RGBQUAD lpPalette; /* Pointer to palette data */
AT_INT i;               /* Index for palette loop */
/* Make a palette in which every color is GREEN */
nErrcount = IG_DIB_palette_alloc(hDIB);
nEntries = IG_DIB_palette_length_get(hDIB);
lpPalette = IG_DIB_palette_pointer_get(hDIB);
for (i = 0; i < nEntries; i++)
{
    lpPalette[i].rgbRed = lpPalette[i].rgbBlue = 0;
    lpPalette[i].rgbGreen = 255;
```

```
}
```

IG_DIB_palette_size_get

Group: DIB Info Services

Declaration:

```
AT_INT ACCUAPI IG_DIB_palette_size_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the size of the DIB's palette, in bytes.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level

Return Value:

DIB palette size, in bytes.

Example:

```
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT paletteSize;     /* Returned size of palette */
paletteSize = IG_DIB_palette_size_get(hDIB);
```

IG_DIB_pixel_array_size_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pixel_array_size_get(
HIGEAR hIGear,
AT_DIMENSION length,
AT_MODE format,
LPAT_DIMENSION lpSize
   );
```

Arguments:

hIGear	HIGEAR handle of image containing pixel data
length	Number of pixels to calculate array size for
format	IG_PIXEL_UNPACKED - all bit depths are unpacked (at least one byte per pixel) IG_PIXEL_PACKED - In legacy mode: 1 and 4 bit images are packed (8 or 2 pixels per byte). In new mode: Only 1 bit images are packed (8 pixels per byte)
lpSize	Returned array size in bytes

Description:

This function returns the number of bytes needed to store an array of pixels. This can be used to allocate storage for use with pixel access functions. This function is similar to IG_DIB_raster_size_get(), but it lets you specify the number of pixels instead of using the number of pixels in an entire raster.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Get # of bytes needed to store half a row */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
AT_DIMENSION w, h;      /* Width and height of image */
AT_DIMENSION nBytes;    /* Size of pixel array in bytes */
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
nErrcount = IG_DIB_pixel_array_size_get(hImage, w / 2,
    IG_PIXEL_UNPACKED, &nBytes);
```

IG_DIB_pixel_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pixel_get ( HIGEAR hIGear,
   AT_PIXPOS nXpos,
   AT_PIXPOS nYpos,
   LPAT_PIXEL lpPixel
   );
```

Arguments:

hIGear	HIGEAR handle of image.
nXpos	X offset (in pixels) from beginning of raster line. First pixel on line is pixel number 0.
nYpos	Raster line number. 0 is top line.
lpPixel	Far pointer to byte at which to store pixel (or to a 3-byte area if a 24-bit pixel)

Description:

This function obtains the pixel at coordinates (nXpos, nYpos), storing it right justified (that is, in the least significant bits) at the location pointed to by lpPixel. If the pixel is 1-bit or 4-bit, the remaining bits of the byte will be set to zeroes.

If the pixel is 24-bit, 3 bytes are returned. These will be in the order Blue-Green-Red (unless you have changed the order of the image bitmap bytes such as by calling function **IG_IP_swap_red_blue()**).

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

TIP: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image*/
AT_PIXELcPixelValue[3];/* 3 bytes in case 24-bit image*/
       /* Get value of the upper-leftmost pixel in image: */
IG_DIB_pixel_get ( hIGear, 0, 0, &cPixelValue[0] );
```

IG_DIB_pixel_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pixel_set ( HIGEAR hIGear,
   AT_PIXPOS nXpos,
   AT_PIXPOS nYpos,
   const LPAT_PIXELlpPixel
   );
```

Arguments:

hIGear	HIGEAR handle of an image.
nXpos	X offset (in pixels) from beginning of raster line. First pixel on line is pixel number 0.
nYpos	Raster line number. 0 is top line.
pPixel	Far pointer to byte containing pixel, or to a 3-byte area if a 24-bit pixel

Description:

This function sets the pixel, at the location pointed to by `lpPixel`, into the image bitmap of image `hIGear` at coordinates (`nXpos`, `nYpos`). The pixel is assumed to have the same number of Bits Per Pixel as the image, and if it is 1 or 4 bits, it is assumed to be right justified (that is, in the least significant bits) at location `lpPixel`. If the image is 24-bit, 3 bytes are transferred. Normally, these bytes will be in the order of Blue-Green-Red (unless the order of the image bitmap bytes has been changed by a call such as **`IG_IP_swap_red_blue()`**).

TIP: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

NOTE: If the image you are modifying is 1-bit, you must convert the image from run-end encoded to a standard DIB, before you can set pixel values. Please see the section Using Pixel Access Functions with ImageGear for details.

Bits Per Pixel:

1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image*/
AT_PIXEL cPixelValue[3];/* 3 bytes in case 24-bit image*/
/* Set upper-leftmost pixel in image, to max pixel value:*/
cPixelValue[0] = cPixelValue[1] = cPixelValue[2] = 255;
IG_DIB_pixel_set ( hIGear, 0, 0, &cPixelValue[0] );
```

IG_DIB_pix_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pix_get(
HIGEAR hIGear,
AT_PIXPOS xpos,
AT_PIXPOS ypos,
HIGPIXEL *lphPixel
   );
```

Arguments:

hIGear	HIGEAR handle of image to get pixel from
xpos	X coordinate (0 to width-1)
ypos	Y coordinate (0 to height-1)
lphPixel	Returns object containing pixel data

Description:

This function gets a pixel from the specified location in the image. The pixel data is contained by a pixel object with handle of type `HIGPIXEL`. This pixel object stores values for each channel in the image.

Bits Per Pixel:

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
```

```
        for (x = 0; x < w / 2; x++)
        {
            nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
            for (c = 0; c < nChannels; c++)
            {
                IG_image_channel_depth_get(hImage, c, &nDepth);
                nDepth = (1 << nDepth) - 1;
                inverted = nDepth - IG_pixel_value_get(hPix, c);
                IG_pixel_value_set(hPix, c, inverted);
            }
            nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
            IG_pixel_delete(hPix);
        }
    nErrcount = IG_save_file(hImage, "test.bmp",
        IG_SAVE_BMP_UNCOMP);
    IG_image_delete(hImage);
```

IG_DIB_pix_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pix_set(
HIGEAR hIGear,
AT_PIXPOS xpos,
AT_PIXPOS ypos,
const HIGPIXEL hPixel
   );
```

Arguments:

hIGear	HIGEAR handle of image to set pixel into
xpos	X coordinate (0 to width-1)
ypos	Y coordinate (0 to height-1)
hPixel	Pixel data to write

Description:

This function sets a pixel at the specified location in the image. The pixel data is contained by a pixel object with handle of type HIGPIXEL. This pixel object stores values for each channel in the image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

IG_DIB_raster_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_raster_get (HIGEAR hIGear,
   AT_PIXPOS nYpos,
   LPAT_PIXEL lpPixel,
   AT_MODE nFormat
   );
```

Arguments:

hIGear	HIGEAR handle of image
nYpos	Raster line number (0 is top line)
lpPixel	Far pointer to first byte of area to receive the raster row of pixel values.
nFormat	A variable of type AT_MODE (see accucnst.h) that tells whether the data being read in packed, unpacked, or RLE-compressed.

Description:

This function obtains an entire horizontal raster line of pixels from the DIB image bitmap. You may first make a call to **IG_DIB_raster_size_get()** in order to determine the size of buffer that you will need to hold the raster data. The format in which the data is returned, nFormat, tells ImageGear whether the data is packed, unpacked, or RLE-compressed. The values that **nFormat** may be set to are: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, and IG_PIXEL_RLE. IG_PIXEL_PACKED gives the storage format of a standard uncompressed DIB, which includes padding to a multiple of 4 bytes length. (If 1-bit or 4-bit, the pixels are packed 8 or 2 to a byte respectively, stored most-significant-bit-first.) 24-bit pixels are returned 3 bytes each, ordered Blue-Green-Red, with the row padded to a multiple of 4 bytes length.

TIP: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR  hIGear;  /* HIGEAR handle of image */
AT_PIXEL cPixelValue[64];/* Needed to hold 500 1-bit pixels*/
AT_PIXPOS nRaster;/* Index for the loop below */
AT_ERRCOUNT nErrcount;/* Hold the returned error count*/
AT_MODE nFormat;
/* Obtain the top raster of a 1-bit image that's 500 pixels
wide:  */
/* The pixels occupy 500/8 = 62.5 bytes.*/
nErrcount = IG_DIB_raster_get ( hIGear, 0, &cPixelValue[0],
IG_PIXEL_PACKED);
/* Make the next 9 rows identical to the top row: */
for ( nRaster = 1; nRaster < 10; nRaster++ )
   nErrcount = IG_DIB_raster_set ( hIGear, nRaster,
&cPixelValue[0], IG_PIXEL_PACKED );
```

IG_DIB_raster_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_raster_set (HIGEAR hIGear,
   AT_PIXPOS nYpos,
   Const LPAT_PIXEL lpPixel,
   AT_MODE nFormat
   );
```

Arguments:

hIGear	HIGEAR handle of an image
nYpos	Raster line number (0 is top line)
lpPixel	Far pointer to first byte of the pixel data to set
nFormat	A variable of type AT_MODE which indicates in which manner the raster data should be stored: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, or IG_PIXEL_RLE.

Description:

This function sets a horizontal raster line of pixels into the DIB image bitmap. nFormat should be set to the same format in which the rest of the data in the DIB is stored. If you choose IG_PIXEL_PACKED, your array of bytes should be padded with zeroes on the right, just as it is in the standard DIB format, such that its length is a multiple of 4.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

TIP: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

NOTE: If the image you are modifying is 1-bit, you must convert the image from run-end encoded to a standard DIB, before you can set pixel values. Please see the section Using Pixel Access Functions with ImageGear for details.

Example:

See the example for function **IG_DIB_raster_get()**.

IG_DIB_raster_size_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_raster_size_get(HIGEAR hIGear,
   AT_MODE nFormat,
   LPAT_DIMENSION lpSize
   );
```

Arguments:

hIGear	HIGEAR handle of the image.
nFormat	Format in which the raster data is stored: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, IG_PIXEL_RLE.
lpSize	A far pointer to a variable in which the size of the raster line (in bytes) is returned.

Description:

This function calculates and returns the number of bytes required to hold a line (raster) from the image. You can use this function to determine the size of a block of memory to allocate before using **IG_DIB_raster_get()**, to avoid data overflow. The returned size will include allocation for buffering at the end of the rasters. (See the section Device-Independent Bitmaps (DIB) for more information on buffering in DIBs.) The nFormat variable determines in what form you would like to store the pixels in. The size will vary according to the storage method.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Also Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

IG_DIB_resolution_get

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_get(
   HIGDIBINFO hDIB,
   AT_RESOLUTION *lpResolution
   );
```

Arguments:

hDIB	DIB info handle
lpResolution	Pointer to struct to copy DIB resolution to

Description:

This function copies resolution from DIB to `lpResolution`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_RESOLUTION res;      /* Returned image resolution */
nErrcount = IG_DIB_resolution_get(hDIB, &res);
```

IG_DIB_resolution_set

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_set(
   HIGDIBINFO hDIB,
   const AT_RESOLUTION *lpResolution
   );
```

Arguments:

hDIB	DIB info handle
lpResolution	Pointer to struct to copy to DIB

Description:

This function copies resolution from `lpResolution` to DIB.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_RESOLUTION res;      /* Image resolution to be set */
/* Set resolution to 300 DPI */
res.units = IG_RESOLUTION_INCHES;
res.xResNumerator = res.yResNumerator = 300;
res.xResDenominator = res.yResDenominator = 1;
nErrcount = IG_DIB_resolution_set(hDIB, &res);
```

IG_DIB_resolution_units_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_units_get(LPAT_MODE
lpnResUnits);
```

Arguments:

lpnResUnits	Pointer to AT_MODE where to return current identifier of resolution units.

Description:

ImageGear allows you to store the DIB resolution - **biXPelsPerMeter** and **biYPelsPerMeter** fields of AT_DIB structure - in different units depending on a global parameter. This function allows you to get the current value of this parameter. Valid values are IG_RESOLUTION_INCHES and IG_RESOLUTION_METERS. The default value is IG_RESOLUTION_METERS.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image */
```

```
AT_MODE nResUnits;

. . .IG_DIB_resolution_units_get( &nResUnits );. . .
```

IG_DIB_resolution_units_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_units_set( AT_MODE
nDIBResUnits);
```

Arguments:

nDIBResUnits	Value of type AT_MODE. Can be either IG_RESOLUTION_INCHES or IG_RESOLUTION_METERS.

Description:

ImageGear allows you to store the DIB resolution (**biXPelsPerMeter** and **biYPelsPerMeter** fields of AT_DIB structure) in different units depending on a global parameter. This function allows you to set the current value of this parameter. Valid values are IG_RESOLUTION_INCHES and IG_RESOLUTION_METERS. The default value is IG_RESOLUTION_METERS.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_MODE nDIBResUnits
nDIBResUnits = IG_RESOLUTION_INCHES
...IG_DIB_resolution_units_set( nDIBResUnits );. . .
```

IG_DIB_row_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_row_get ( HIGEAR hIGear,
   AT_PIXPOS nXpos,
   AT_PIXPOS nYpos,
   AT_DIMENSION nLength,
   LPVOID lpPixel,
```

```
AT_MODE nFormat
);
```

Arguments:

hIGear	HIGEAR handle of image
nXpos	X offset (in pixels) from beginning of the raster line. First pixel on line is pixel number 0. Specify as -1 to obtain the entire raster line of pixels.
nYpos	Raster line number (0 is top line)
nLength	Number of consecutive pixels to obtain (enitre raster line if nXpos = -1)
lpPixel	Far pointer to first byte of your area, at which the pixels obtained are to be stored
nFormat	Format in which the raster data is read: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, IG_PIXEL_RLE.

Description:

This function obtains a consecutive row of **nLength** pixels beginning at coordinates (nXpos, xYpos) in the image bitmap. If nFormat is set to IG_PIXEL_UNPACKED, and the image is 1 or 4-bit, the pixels obtained are stored right justified, one to a byte, beginning at your byte pointed to by lpPixel. The unused bits of the bytes are set to zero. If the pixels are 24-bit, 3 bytes per pixel are returned, ordered Blue-Green-Red. A total of **nLength** pixels is transferred. (See the section Device-Independent Bitmaps (DIB)Understanding Bitmap Images for more details on pixel storage in DIBs.)

TIP: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

Bits Per Pixel:

1, 4, 8, 9-16-bit gray level, 24. 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR  hIGear;/* HIGEAR handle of image  */
AT_PIXEL cPixelValue[300];/* To hold 100 pixels, in case 24-
bit*/
AT_PIXPOS nRow;/* Index for the loop below  */
AT_DIMENSION nRowLen;/* How much of row to copy   */
AT_ERRCOUNT nErrcount;/* Holds the returned error count */
/* Obtain leftmost 100 pixels of top raster of image:  */
nRowLen = 100;
nErrcount = IG_DIB_row_get ( hIGear, 0, 0, nRowLen,
&cPixelValue[0], IG_PIXEL_UNPACKED );
```

```
/* Make leftmost 100 pixels of next 9 rows identical:
*/
for ( nRow = 1; nRow < 10; nRow++ )
   nErrcount = IG_DIB_row_set (hIGear, 0, nRow,
      nRowLen, cPixelValue, IG_PIXEL_UNPACKED);
```

IG_DIB_row_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_row_set ( HIGEAR hIGear,
   AT_PIXPOS nXpos,
   AT_PIXPOS nYpos,
   AT_DIMENSION nLength,
   const LPVOID lpPixel,
   AT_MODE nFormat
   );
```

Arguments:

hIGear	HIGEAR handle of image
nXpos	X offset (in pixels) from beginning of raster line. First pixel on line is pixel number 0
nYpos	Raster line number. 0 is top line
nLength	Number of consecutive pixels to transfe
lpPixel	Far pointer to byte at which the pixels to transfer begin
nFormat	Format in which the raster data is stored: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, IG_PIXEL_RLE.

Description:

This function writes a consecutive row of **nLength** pixels that begin at lpPixel, into image hIGear's DIB image bitmap. The row is written into raster line nYpos, beginning at pixel offset nXpos. If nFormat is set to IG_PIXEL_UNPACKED, and the image is 1-bit or 4-bit, your pixels to be transferred should be one to a byte, right justified (that is, in the least significant bits). If a 24-bit image, each pixel should occupy 3 bytes, ordered Blue-Green-Red. (See the section Device-Independent Bitmaps (DIB) for more details on pixel storage in DIBs.)

(If (**nXpos + nLength**) is greater than the width of the image as indicated in the DIB header, an error will result.

TIP: ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

NOTE: If the image you are modifying is 1-bit, you will probably need to convert the image from run-end encoded to a standard DIB, before you can set pixel values. Please see the section Using Pixel Access Functions with ImageGear Using ImageGearfor details.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

See the example for function **`IG_DIB_row_get()`**.

IG_DIB_width_get

Group: DIB Info Services

Declaration:

```
AT_DIMENSION ACCUAPI IG_DIB_width_get(
   HIGDIBINFO hDIB
   );
```

Arguments:

hDIB	DIB info handle

Description:

This function returns the width of the image.

Bits Per Pixel: All

Return Value:

Width of the image.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_DIMENSION width;     /* Returned height of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
```

```
width = IG_DIB_width_get(hDIBInfo);
```

IG_display_animation_delay_get

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_display_animation_delay_get ( HIGEAR
hIGear,
   LPUINT lpDelay
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpDelay	Far pointer to a UINT variable to receive current animation delay setting for this image

Description:

This function obtains the animation delay setting for image hIGear, as set by **IG_display_animation_delay_set()**. ImageGear does not provide functions for creating animation. This function is for getting this value from the HIGEAR. You will need to write your own code to display the images in succession (animate).

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;/* HIGEAR handle of image    */
UINT    nDelay;/* Will hold returned Delay setting     */
IG_display_animation_delay_get ( hIGear, &nDelay );
```

IG_display_animation_delay_set

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_display_animation_delay_set (HIGEAR
hIGear,
   UINT   nDelay
   );
```

Arguments:

hIGear	HIGEAR handle of image
nDelay	Animation delay, in milliseconds. A setting of 100 would mean 10 frames per second.

Description:

This function sets the animation delay for image `hIGear`. (At present, this quantity is meaningful only with GIF format files.) ImageGear does not provide functions for creating animation. This function is for setting this value in the **`HIGEAR`**. You will need to write your own code to display the images in succession (animate).

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;  /* HIGEAR handle of image*/
/* Set for 5 frames per second: */
IG_display_animation_delay_set ( hIGear, 200 );
```

- `IG_DISPLAY_OPTION_OFFSCREEN_WIDTH/ IG_DISPLAY_OPTION_OFFSCREEN_HEIGHT`**:** Default values (0, 0) means that GetDeviceCaps(hDC,HORZRES) / GetDeviceCaps(hDC,VERTRES) will be used, respectively. These options specify the size of offscreen drawing surface. Set it to the size of a single monitor. In case of a dual monitor system you need to set `IG_DISPLAY_OPTION_OFFSCREEN_WIDTH / IG_DISPLAY_OPTION_OFFSCREEN_HEIGHT` to the total size of your dual monitor screen.

IG_dspl_antialias_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwGrpID,
[OUT]  LPAT_MODE lpnAliasFlags,
[OUT]  LPUINT lpnThreshold
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to get option.
lpnAliasFlags	Pointer to where `AliasMode` is to be received. If `NULL`, then this parameter is ignored.
lpnThreshold	Pointer to where `ThresholdValue` is to be received. If `NULL`, then this parameter is ignored.

Description:

This function returns the current anti-alias settings. Possible values are listed in the description of the **`IG_dspl_antialias_set()`** function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier */
AT_MODE nAliasFlags; /* alias flags  */
UINT   nThreshold;  /* alias threshold */

 …
IG_dspl_antialias_get( hIGear, nGrpID, &nAliasFlags,
&nThreshold );
 …
```

IG_dspl_antialias_get_ex

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_get_ex(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwGrpID,
[OUT]  LPAT_MODE lpnAliasFlags,
[OUT]  LPUINT lpnThreshold
[OUT] LPUINT lpnQuality
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to get option.
lpnAliasFlags	Pointer to where `AliasMode` is to be received. If `NULL`, then this parameter is ignored.
lpnThreshold	Pointer to where `ThresholdValue` is to be received. If `NULL`, then this parameter is ignored.
lpnQuality	Pointer to where color antialiasing quality value is to be received. If NULL, then this parameter is ignored.

Description:

This function returns the current anti-alias settings. Possible values are listed in the description of the **`IG_dspl_antialias_set_ex()`** function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier */
AT_MODE nAliasFlags; /* alias flags  */
UINT   nThreshold;  /* alias threshold */
UINT nQuality /*alias quality */
 …
IG_dspl_antialias_get_ex( hIGear, nGrpID, &nAliasFlags,
&nThreshold, &nQuality);
 …
```

Sample: **None**

IG_dspl_antialias_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nAliasFlags,
[IN]   INT    nThreshold
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set option.

nAliasFlags	New value of `AliasMode` to set. Possible value is one from this list:
	`IG_DSPL_ANTIALIAS_NONE`
	`IG_DSPL_ANTIALIAS_SCALE_TO_GRAY`
	`IG_DSPL_ANTIALIAS_PRESERVE_BLACK`
	`IG_DSPL_ANTIALIAS_PRESERVE_WHITE`
	with OR combination of one additional flag
	`IG_DSPL_ANTIALIAS_SUBSAMPLE`

nThreshold	Specifies display aliasing threshold level (AliasThreshold). Please see Dithering, Anti-aliasing, and Palette Handling.

Description:

This function sets new anti-alias settings.

Bits Per Pixel: 1

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier */
 …
/* sets scale to gray algorithm with subsampling  */
IG_dspl_antialias_set( hIGear, nGrpID,
IG_DSPL_ANTIALIAS_SCALE_TO_GRAY| IG_DSPL_ANTIALIAS_SUBSAMPLE,
50 );
 …
```

IG_dspl_antialias_set_ex

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_set_ex(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nAliasFlags,
[IN]   INT    nThreshold,
```

```
[IN] INT nColorQuality
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set option.
nAliasFlags	New value of AliasMode to set. It is an OR combination of following elements: ■ Scale-Up interpolation (resampling) ■ Scale-Down interpolation (antialiasing) For 1 bit images antialiasing element is a one of constants: ■ `IG_DSPL_ANTIALIAS_NONE` ■ `IG_DSPL_ANTIALIAS_SCALE_TO_GRAY` ■ `IG_DSPL_ANTIALIAS_PRESERVE_BLACK` ■ `IG_DSPL_ANTIALIAS_PRESERVE_WHITE` with OR combination of one additional flag `IG_DSPL_ANTIALIAS_SUBSAMPLE` Antialising for all non-1bit color spaces is activated with the `IG_DSPL_ANTIALIAS_COLOR` flag. Scale-Up interpolation for all non-1bit color spaces is specified by the `IG_DSPL_ANTIALIAS_RESAMPLE_BILINE` flag.
nThreshold	It indicates the threshold value of the amount of selected color to include for `IG_DSPL_ANTIALIAS_PRESERVE_XXXX` scale-down interpolations. Please see Dithering, Anti-aliasing, and Palette Handling.
nColorQuality	Specifies quality of color scale-down interpolation (antialiasing). Valid values are from 0 to 100 and specify the percent of image pixels taken into account to produce one display pixel. Zero value means automatic quality choice according with image resolution and display settings.

Description:

This function sets new anti-alias settings.

Bits Per Pixel: 1

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier */
 …
/* sets scale to gray algorithm with subsampling  */
IG_dspl_antialias_set_ex( hIGear, nGrpID,
IG_DSPL_ANTIALIAS_SCALE_TO_GRAY| IG_DSPL_ANTIALIAS_SUBSAMPLE,
50, 80);
```

...

Sample: None

IG_dspl_artx_image_draw

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_artx_image_draw(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   LPVOID hArtXPage,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[OUT]  LPAT_RECTANGLE lpActualDeviceRect
);
```

Arguments:

hIGear	ImageGear handle from which to get options and pixel data
dwGrpID	Identifier of group from which to get display options
hArtXPage	ArtX page to display.
hWnd	Handle of window where the image is to be displayed. Can be `NULL`, but in this case some display options may be calculated incorrectly if they are not set explicitly.
hDC	Handle of the device context on which to draw the image. Can be `NULL`, and in this case no actual drawing is performed but all parameters are calculated and updated. This may be useful in calculating `DisplayedImageRect` in the next parameter of this function.
lpActualDeviceRect	Pointer to the rectangle in which to copy `DisplayedImageRect`. This rectangle is calculated no matter whether `hDC=NULL` or not. If the value is `NULL`, then this parameter is ignored.

Description:

This function acts like **`IG_dspl_image_draw()`** but takes one extra parameter `hArtXPage`.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_dspl_artx_image_print

```
[IN]   LPVOID hArtXPage,
```

IG_dspl_artx_page_print

```
[IN]   LPVOID hArtXPage,
```

IG_dspl_background_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_background_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPAT_MODElpnBkMode,
[OUT]  LPAT_RGBlpBkColor,
[OUT]  HBITMAP FAR*lphBrush
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of the group to use.
lpnBkMode	Pointer to where `BkMode` options are returned. If `NULL`, then this parameter is ignored.
lpBkColor	Pointer to where `BkColor` option is returned. If `NULL`, then this parameter is ignored.
lphBrush	Pointer to where `BkBrush` option is returned. If `NULL`, then this parameter is ignored.

Description:

This function returns the background settings' current values. Possible values are listed in the description of function **`IG_dspl_background_set()`**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image  */
DWORD  nGrpID;      /* display group identifier */
AT_MODE nBkMode;    /* backgound mode  */
AT_RGB BkColor;    /* background color  */
HBITMAP hBrush;     /* background mask  */
 …
IG_dspl_background_get( hIGear, nGrpID, &nBkMode, &BkColor,
&hBrush );
 …
```

IG_dspl_background_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_background_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nBkMode,
[IN] const LPAT_RGB lpBkColor,
[IN]   HBITMAP hBrush
);
```

Arguments:

hIGear	ImageGear handle of image
dwGrpID	Identifier of the group in which to set options.

nBkMode	`IG_DSPL_BACKGROUND_NONE` or a combination of the following flags:
	`IG_DSPL_BACKGROUND_UNDER_IMAGE`
	`IG_DSPL_BACKGROUND_BEYOND_IMAGE`

lpBkColor	New value of `BkColor` to set. If `NULL`, then this parameter is ignored.
hBrush	New value of `BkBrush` option to set. Please note that the old bitmap is not deleted and the application code is responsible for removing the old value of this option.

Description:

This function sets the background settings.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
DWORD  nGrpID;  /* display group identifier  */
 …
/* disable background under image and beyond image  */
IG_dspl_background_set( hIGear, nGrpID,
IG_DSPL_BACKGROUND_NONE, NULL, NULL );
 …
```

IG_dspl_DDB_create

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_DDB_create(
[IN]   HIGEAR    hIGear,
[IN]   DWORD     dwGrpID,
[IN]   HDC       hDC,
[IN]   AT_DIMENSION nWidth,
[IN]   AT_DIMENSION nHeight,
[IN]   BOOL      bExport,
[OUT]  HBITMAP FAR* lphBitmap,
[OUT]  HPALETTE FAR* lphPalette
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which the options are stored
hDC	Handle of device context that the DDB should be compatible with. If NULL then the DDB will be compatible with the desktop's device context.
nWidth	Width of DDB that is to be created.
nHeight	Height of DDB that is to be created.
bExport	Boolean parameter which specifies whether to delete the source image or not. If TRUE then the hIGear image will be deleted after the DDB is created, but if FALSE then the image is left unchanged.
lphBitmap	Pointer to where to return the DDB object.
lphPalette	Pointer to where to return the palette handle for this DDB.

Description:

This function creates a DDB of the image. It always uses all the display options specified by dwGrpID group and assumes that the output device has a color format compatible with the **hDC**, but the client area of the output device is a rectangle of **nWidth x nHeight** size.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier */
HBITMAP hBitmap;    /* handle of bitmap  */
HPALETTE hPalette;   /* handle of palette */

 …
case ID_IMAGE_EXPORT:
    …
IG_dspl_DDB_create( hIGear, nGrpID, NULL, 640, 480, TRUE,
&hBitmap, &hPalette );
    …
   break;
 …
```

IG_dspl_DDB_draw

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI IG_dspl_DDB_draw(
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[IN]   HBITMAP hBitmap,
[IN]   HPALETTE hPalette,
[IN]   AT_PIXPOS x,
[IN]   AT_PIXPOS y
);
```

Arguments:

hWnd	Handle of window to be used. Can be NULL if not available.
hDC	Handle of device context in which to draw.
hBitmap	Device dependent bitmap in OS-dependent format in which to draw.
hPalette	Handle of palette which should be realized before drawing.
x	Horizontal position of left top corner of bitmap on the destination device.
y	Vertical position of left top corner of bitmap on the destination device.

Description:

This function draws a DDB image on the device. If `hPalette!=NULL` then the palette is selected and realized first.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HBITMAP hBitmap;     /* bitmap to draw  */
HPALETTE hPalette;    /* associated palette handle  */
 …
case WM_PAINT:
   BeginPaint(hWnd, &ps);

   IG_dspl_DDB_draw( hWnd, ps.hdc, hBitmap, hPalette, 0, 0 );
   EndPaint(hWnd, &ps);
   break;
 …
```

IG_dspl_DDB_import

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_DDB_import(
[IN]   HDC    hDC,
[IN]   HBITMAP hBitmap,
[IN]   HPALETTE hPalette,
[OUT]  LPHIGEAR lphIGear
);
```

Arguments:

hDC	Handle of device context that the DDB is compatible with. If NULL then it is assumed that the DDB is compatible with the desktop's device context.
hBitmap	Handle of DDB object that image should be created from.
hPalette	Handle of palette that should be realized before drawing the DDB.
lphIGear	Pointer to where to return the newly allocated image handle.

Description:

This function creates an image from the DDB object. If the color format of the DDB is not supported directly by ImageGear then it is converted to the nearest supported format with a minimal possible loss of color quality.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HBITMAP hBitmap;      /* handle of bitmap  */
HPALETTE hPalette;     /* handle of palette  */
HIGEAR hIGear;       /* HIGEAR handle of image  */

 …
case ID_IMAGE_IMPORT:
    …
IG_dspl_DDB_import( NULL, hBitmap, hPalette, &hIGear );
    …
   break;
 …
```

IG_dspl_device_to_image

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_device_to_image(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[IN/OUT]LPAT_POINT lpPoint,
[IN]   UINT   nCount
);
```

Arguments:

hIGear	ImageGear handle of image.
DwGrpID	Identifier of the group from which to get the display options.
HWnd	Handle of the window where the image is drawn.
HDC	Handle of the device context used for drawing. This can be `NULL`, but if a calculation is necessary for a printer device context, then you should provide the real value.
LpPoint	Pointer to an array of points that should be translated.
NCount	Number of elements in the `lpPoint` array.

Description:

This function translates an array of points from device coordinates to image coordinates. It takes into account all display parameters including orientation and current scrolling position.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier  */
AT_POINT p[2];       /* array of point to translate  */
RECT   rc;

 …
/* calculate coordinates in image coordinate space of current
client rectangle of the window */
GetClientRect( hWnd, &rc );
p[0].x = rc.left;
p[0].y = rc.top;
p[1].x = rc.right;
p[1].y = rc.bottom;

IG_dspl_device_to_image( hIGear, nGrpID, hWnd, NULL, p, 2 );
 …
```

IG_dspl_device_to_image_d

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_device_to_image_d(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[IN/OUT]LPAT_DPOINT lpPoint,
[IN]   UINT   nCount
);
```

Arguments:

hIGear	ImageGear handle of image.
DwGrpID	Identifier of the group from which to get display options.
HWnd	Handle of the window from which the image is drawn.
HDC	Handle of the device context used for drawing. This can be `NULL`, but if a calculation is necessary for printer device contex,t then you should provide the real value.
LpPoint	Pointer to an array of points in DOUBLE float-point format that should be translated.
NCount	Number of elements in the `lpPoint` array.

Description:

This function translates an array of points in **DOUBLE float-points format** from the device coordinates to the image coordinates. It takes into account all display parameters, including the orientation and current scrolling position.

NOTE: See also the function **`IG_dspl_image_to_device_d()`**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_dspl_dithering_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_dithering_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPAT_MODE lpnDitherFlags
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to get option.
lpnDitherFlags	Pointer where `DitherMode` is to be received.

Description:

This function returns the current dithering flags. All possible values are listed in the description of function **`IG_dspl_dithering_set()`**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
DWORD  nGrpID;    /* display group identifier */
AT_MODE nDitherFlags; /* dither flags */

 …
IG_dspl_dithering_get( hIGear, nGrpID, &nDitherFlags );
 …
```

IG_dspl_dithering_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI   IG_dspl_dithering_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nDitherFlags
```

```
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to set dithering.

nDitherFlags	New value of `DitherMode` to set. Possible values include:
	`IG_DSPL_DITHER_AUTO`
	`IG_DSPL_DITHER_TO_8BPP`
	`IG_DSPL_DITHER_TO_4BPP`
	`IG_DSPL_DITHER_TO_1BPP`
	`IG_DSPL_DITHER_NONE`
	with OR combination from two flags
	`IG_DSPL_DITHER_FIXED_PALETTE`
	`IG_DSPL_DITHER_NETSCAPE_PALETTE`

Description:

This function sets new dithering options.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEARhIGear;    /* HIGEAR handle of image */
DWORDnGrpID;    /* display group identifier  */
 …
/* if device is palette based then dither to fixed palette */
IG_dspl_dithering_set( hIGear, nGrpID, IG_DSPL_DITHER_AUTO|
IG_DSPL_DITHER_FIXED_PALETTE );
 …
```

IG_dspl_foreground_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI   IG_dspl_foreground_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwGrpID,
[OUT]  LPAT_RGB lpFrColor
);
```

Arguments:

hIGear	The ImageGear handle of an image.
dwGrpID	Identifier of the group to use.
lpFrColor	Pointer to where the `FrColor` option is returned. If `NULL`, then this parameter is ignored.

Description:

This function returns the foreground color.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image   */
DWORD nGrpID;        /* display group identifier  */
AT_RGB FrColor;       /* foreground color  */
 …
IG_dspl_foreground_get( hIGear, nGrpID, &FrColor );
 …
```

IG_dspl_foreground_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_foreground_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwGrpID,
[IN] const LPAT_RGB lpFrColor
);
```

Arguments:

hIGear	ImageGear handle of an image
dwGrpID	Identifier of the group in which to set options.
lpFrColor	The new value of `FrColor` to set. If `NULL`, then this parameter is ignored.

Description:

This function sets the foreground color.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image   */
DWORD  nGrpID;      /* display group identifier  */
AT_RGB FrColor;
 …
FrColor.r = FrColor.g = FrColor.b;
IG_dspl_foreground_set( hIGear, nGrpID, &FrColor );
…
```

IG_dspl_free_grp_id_get

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_free_grp_id_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwMin,
[OUT]  LPDWORD lpdwFreeGrpId
);
```

Arguments:

hIGear	ImageGear handle of image.
dwMin	Minimum value that IG should start to search from.
lpdwFreeGrpId	Pointer to where to return free group identifier.

Description:

This function searches and returns the first free unused group identifier. It also sets a flag so the returned group is marked as used and the next call to this function with the same parameters returns another group id. To mark a group as unused and reset it to the default values call function **IG_dspl_grp_reset()**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of image   */
DWORD nGrpID;   /* display group identifier  */
 …
IG_dspl_free_grp_id_get( hIGear, 0, &nGrpID );
 …
```

IG_dspl_gamma_correction_LUT_build

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_gamma_correction_LUT_build(
[IN]   DOUBLE dblContrast,
[IN]   DOUBLE dblBrightness,
[IN]   DOUBLE dblGamma,
[OUT]  LPBYTE lpLUT
);
```

Arguments:

dblContrast	Contrast value to use in calculations. You can use any value.
dblBrightness	Brightness value to use in calculations. Must be within the –255.0 to +255.0 range.
dblGamma	Gamma value to use in calculations. Must be greater than 0.0, however the most useful values are in the range from 1.8 to 2.2.
lpLUT	Pointer to 256 byte array in which to calculate the lookup table.

Description:

This function builds lookup tables from given contrast, brightness and gamma values.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
BYTE lut[256];    /* lookup array  */
 …
IG_dspl_gamma_correction_LUT_build( 2.0, 120.0, 2.0, lut );
```

IG_dspl_gamma_correction_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_gamma_correction_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwGrpID,
[IN]   AT_MODE nFlags,
[IN]   DOUBLE dblContrast,
[IN]   DOUBLE dblBrightness,
[IN]   DOUBLE dblGamma
);
```

Arguments:

hIGear	ImageGear handle of image
dwGrpID	Identifier of group in which to set lookup tables.
nFlags	Specify which lookup tables to set. Possible values are 0 or a combination of these flags: `IG_DSPL_R_CHANNEL` – if this flag is set then the `RedLut` option is to be set. `IG_DSPL_G_CHANNEL` – if this flag is set then the `GreenLut` option is to be set. `IG_DSPL_B_CHANNEL` – if this flag is set then the `BlueLut` option is to be set. The constant `IG_DSPL_ALL_CHANNELS` is defined for convenience and can be used to set all three options. `#define IG_DSPL_ALL_CHANNELS (IG_DSPL_R_CHANNEL|IG_DSPL_G_CHANNEL|IG_DSPL_B_CHANNEL)`
dblContrast	Contrast value to use in calculations. You can use any value.
dblBrightness	Brightness value to use in calculations. Must be in the range from –255.0 to +255.0.
dblGamma	Gamma value to use in calculations. Must be greater than 0.0; however the most useful values are in the range from 1.8 to 2.2.

Description:

This function takes contrast (`dblContrast`), brightness (`dblBrightness`) and gamma (`dblGamma`) parameters and calculates lookup tables accordingly and sets them into the corresponding options.

All lookup tables specified in the `nFlags` parameter are initialized with the same value based on the `dblContrast`, `dblBrightness`, and `dblGamma` values.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
DWORD  nGrpID;    /* display group identifier */
 …
IG_dspl_gamma_correction_set( hIGear, nGrpID,
IG_DSPL_ALL_CHANNELS, 2.0, 120.0, 2.0 );
 …
```

IG_dspl_grayscale_LUT_copy_get

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_grayscale_LUT_copy_get(
    HIGEAR hIGear,
    DWORD dwGrpID,
    HIGLUT *lpLUT
);
```

Arguments:

hIGear	ImageGear handle.
dwGrpID	Display group ID.
lpLUT	New LUT handle.

Description:

This function returns a copy of display grayscale LUT, if it exists in the specified display settings group.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

None

IG_dspl_grayscale_LUT_exists

Group: Display

Declaration:

```
AT_BOOL ACCUAPI IG_dspl_grayscale_LUT_exists(
    HIGEAR hIGear,
    DWORD dwGrpID
);
```

Arguments:

hIGear	HIGEAR handle.
dwGrpID	Display group ID.

Description:

This function checks whether the display settings group has a grayscale LUT attached.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

None

IG_dspl_grayscale_LUT_update_from

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_grayscale_LUT_update_from(
   HIGEAR hIGear,
   DWORD dwGrpID,
   HIGLUT lut
);
```

Arguments:

hIGear	HIGEAR handle.
dwGrpID	Display group ID.
lut	LUT handle.

Description:

This function updates (creates if not present) a grayscale LUT for the image display.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

IG_dspl_grp_reset

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_grp_reset(
[IN]   HIGEAR hIGear,
[IN]   DWORD dwGrpID
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group to reset.

Description:

This function resets all the options of the specified group to its default values. Please note the application code is responsible for freeing memory allocated for options such as `TranspMask` and `BkBrush`. This function does not free it.

This function also removes the flag that marks this group as “used.” It may be set again by any display function that specifies using this group identifier.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image */
DWORD nGrpID;/* display group identifier*/
 …
IG_dspl_grp_reset( hIGear, nGrpID );
 …
```

IG_dspl_image_calc

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_calc(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[OUT]  LPAT_RECTANGLE lpActualDeviceRect
);
```

Arguments:

hIGear	ImageGear handle from which to get options and pixel data
dwGrpID	Identifier of group from which to get display options
hWnd	Handle of window where image is to be displayed. Can be `NULL`, but in this case some display options may be calculated incorrectly if they are not set explicitly.
hDC	Handle of the device context where image is to be drawn.
lpActualDeviceRect	Pointer to the rectangle in which to copy `DisplayedImageRect`. This rectangle is calculated no matter whether `hDC=NULL` or not. If `NULL`, then this parameter is ignored.

Description:

This function calculates `DisplayedImageRect`, which is the exact rectangle in which the image is to be scaled. All options for drawing the image are taken from the `dwGrpID` group. This function does not perform any drawing, whether `hDC` is `NULL` or not.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR    hIGear;/* HIGEAR handle of image */
DWORD     nGrpID;/* Display group identifier*/
AT_RECTANGLE DisplayedRect;
…
case WM_PAINT:
   BeginPaint(hWnd, &ps);

   if (IG_image_is_valid(hIGear))
```

```
        IG_dspl_image_calc( hIGear, nGrpID, hWnd, ps.hdc,
&DisplayedRect );
    EndPaint(hWnd, &ps);
    break;
```

IG_dspl_image_draw

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_draw(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[OUT]  LPAT_RECTANGLE lpActualDeviceRect
);
```

Arguments:

hIGear	ImageGear handle from which to get options and pixel data
dwGrpID	Identifier of group from which to get display options
hWnd	Handle of window where the image is to be displayed. Can be `NULL`, but in this case some display options may be calculated incorrectly if they are not set explicitly.
hDC	Handle of the device context on which to draw the image. Can be `NULL`, and in this case no actual drawing is performed but all parameters are calculated and updated. This may be useful in calculating `DisplayedImageRect` in the next parameter of this function.
lpActualDeviceRect	Pointer to the rectangle in which to copy `DisplayedImageRect`. This rectangle is calculated no matter whether `hDC=NULL` or not. If the value is `NULL`, then this parameter is ignored.

Description:

This function draws an image onto a destination device context. All options regarding how to draw the image are taken from the `dwGrpID` group. This function can also be used for the actual device rectangle calculation. If `hDC=NULL` then all parameters are calculated and updated but no actual drawing is performed.

NOTE: If you are using ImageGear ARTX component in your application please use **`IG_dspl_artx_image_draw()`** function.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image */
DWORD  nGrpID;/* Display group identifier*/
…
case WM_PAINT:
   BeginPaint(hWnd, &ps);

   if (IG_image_is_valid(hIGear))
      IG_dspl_image_draw( hIGear, nGrpID, hWnd, ps.hdc, NULL );
   EndPaint(hWnd, &ps);
   break;
```

NOTE: If you are using ImageGear ARTX component in your application please use **`IG_dspl_artx_image_print()`** function.

IG_dspl_image_to_device

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_to_device(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[IN/OUT]LPAT_POINT lpPoint,
[IN]   UINT   nCount
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to get display options.
hWnd	Handle of a window where the image is drawn.
hDC	Handle of the device context used for drawing. Can be `NULL`, but if it is necessary to perform a calculation for the printer device context, then a real value should be provided.
lpPoint	Pointer to an array of points that should be translated.
nCount	Number of elements in `lpPoint` array.

Description:

This function translates an array of pointers from image coordinates into device coordinates. It takes into account all display parameters including orientation and current scrolling position.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image */
DWORD  nGrpID;/* display group identifier*/
AT_POINT p[2];/* array of point to translate */
AT_RECTANGLE ImageRect;/* image rectangle*/

 …
/* calculates device coordinates of current image rectangle */
IG_dspl_layout_get( hIGear, nGrpID, &ImageRect, NULL, NULL,
NULL, NULL, NULL, NULL );
p[0].x = ImageRect.x;
p[0].y = ImageRect.y;
p[1].x = ImageRect.x + ImageRect.width - 1;
p[1].y = ImageRect.y + ImageRect.height - 1;

IG_dspl_image_to_device( hIGear, nGrpID, hWnd, NULL, p, 2 );
 …
```

IG_dspl_image_to_device_d

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_to_device_d(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[IN/OUT]LPAT_DPOINT lpPoint,
[IN]   UINT   nCount
);
```

Arguments:

hIGear	ImageGear handle of an image.
dwGrpID	Identifier of the group from which to get display options.
hWnd	Handle of the window where the image is drawn.
hDC	Handle of the device context used for drawing. Can be NULL, but if it is necessary to perform a calculation for the printer device context, then a real value should be provided.
lpPoint	Pointer to an array of points in **DOUBLE float-point format** that should be translated.
nCount	Number of elements in the `lpPoint` array.

Description:

This function translates an array of points in **DOUBLE float-point format** from the image coordinates into device coordinates. It takes into account all display parameters, including orientation and current scrolling position.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

NOTE: See also **`IG_dspl_device_to_image_d()`** function.

IG_dspl_image_wipe

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_image_wipe(
[IN]   HIGEAR hIGearBefore,
[IN]   DWORD  dwGrpBefore,
[IN]   HIGEAR hIGearAfter,
[IN]   DWORD  dwGrpAfter,
[IN]   HWND   hWnd,
[IN]   AT_MODE nWipeStyle,
[IN]   LONG   nGranularity,
[IN]   LONG   lTime
);
```

Arguments:

hIGearBefore	HIGEAR handle of an image in the window before the wipe.
dwGrpBefore	Display group identifier used for hIGearBefore.
hIGearAfter	HIGEAR handle of an image to be in window after the wipe.
dwGrpAfter	Display group identifier used for hIGearAfter.
hWnd	Window handle.
nWipeStyle	Transition style, such as IG_WIPE_LEFTTORIGHT. See below. (Check updates to accucnst.h for new wipe styles.)
nGranularity	Size of each square region, in pixels. A typical value is 5.
lTime	The time, in milliseconds, between wipes

Description:

This function changes the image in window **hWnd** from the hIGearBefore image to the hIGearAfter image. The transition is accomplished according to the nWipeStyle style. Before you call this function, the first image should already be in the window. This function will set the wipe style, and then generate a WM_PAINT message. The **IG_dspl_image_draw()** call while processing this **WM_PAINT** message will perform the transition to the new image. nWipeStyle Constants include:

IG_WIPE_LEFTTORIGHT	Left-to-Right Wipe.
IG_WIPE_RIGHTTOLEFT	Right-To-Left Wipe.
IG_WIPE_UP_TO_DOWN	Up-to-Down Wipe.
IG_WIPE_DOWN_TO_UP	Down-to-Up Wipe.

IG_WIPE_SPARKLE	Sparkle Transition.
IG_WIPE_ULTOLRDIAG	Upper Left to Lower Right wipe.
IG_WIPE_LRTOULDIAG	Lower Right to Upper Left wipe.
IG_WIPE_URTOLLDIAG	Upper Right to Lower Left wipe.
IG_WIPE_LLTOURDIAG	Lower Left to Upper Right wipe.
IG_WIPE_CLOCK	Clockwise wipe.
IG_WIPE_SPARKLE_CLOCK	Clockwise wipe with sparkles.
IG_WIPE_DOUBLE_CLOCK	Two simultaneous clockwise wipes, 180° apart.
IG_WIPE_SLIDE_RIGHT	New image slides in from the left.
IG_WIPE_SLIDE_LEFT	New image slides in from the right.
IG_WIPE_SLIDE_UP	New image slides in from the down.
IG_WIPE_SLIDE_DOWN	New image slides in from the up.
IG_WIPE_RANDOM_BARS_DOWN	Vertical bars of old image fall to reveal new image.
IG_WIPE_RAIN	Vertical lines of new image cover over old, like paint running down the side of a bucket.
IG_WIPE_BOOK	Bookwisw wipe.
IG_WIPE_ROLL	Old image rolls in from right to left.
IG_WIPE_UNROLL	New image rolls out from left to right.
IG_WIPE_EXPAND_PROPORTIONAL	New image expands from the center of old image in diagonal directions.
IG_WIPE_EXPAND_HORIZONTAL	New image expands from the center of old image in horizontal directions.
IG_WIPE_EXPAND_VERTICAL	New image expands from the center of old image in vertical directions.
IG_WIPE_STRIPS_HORIZONTAL	New image appeares as expanding horizontal strips.
IG_WIPE_STRIPS_VERTICAL	New image appeares as expanding vertical strips.
IG_WIPE_CELLS	New image appeares as expanding square cells.
IG_WIPE_BALL	New image appeares as tracks of spirally moving balls.
IG_WIPE_GEARS	New image appeares as tracks of moving ImageGear's icons.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGearBefore;   /* Handle of image already being
displayed */
HIGEAR  hIGearAfter;    /* Handle of image to replace it  */
HWND    hWnd;            /* Handle of the window  */
AT_ERRCOUNT nErrcount;     /* Returned count of errors  */
/* Perform a “sparkle” wipe, changing 16 x 16 pixel areas in
each step:*/
nErrcount = IG_dspl_image_wipe ( hIGearBefore, IG_GRP_DEFAULT,
hIGearAfter, IG_GRP_DEFAULT, hWnd, IG_WIPE_SPARKLE, 16, 5 );
/* See sample application WIPES for a complete example */
```

IG_dspl_layout_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_layout_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPAT_RECTANGLE lpImageRect,
[OUT]  LPAT_RECTANGLE lpDeviceRect,
[OUT]  LPAT_RECTANGLE lpClipRect,
[OUT]  LPAT_MODE lpnFitMode,
[OUT]  LPAT_MODE lpnAlignMode,
[OUT]  LPAT_MODE lpnAspectMode,
[OUT]  LPDOUBLE lpdAspectValue
);
```

Arguments:

hIGear	Image handle from which to search the option group.
dwGrpID	Identifier of group which to get layout options.
lpImageRect	Pointer to the rectangle in which to copy the value of `ImageRect` option. If `NULL` then it is ignored. If `ImageRect` is not set then an empty rectangle is returned.
lpDeviceRect	Pointer to the rectangle in which to copy `DeviceRect`. If `NULL`, then the parameter is ignored. If an empty rectangle is returned then it is not set and it will be calculated every time from the destination device.
lpClipRect	Pointer to the rectangle in which to copy `ClipRect`. If `NULL`, then the parameter is ignored. If an empty rectangle returned then it is not set and it will be calculated every time from the destination device.
lpnFitMode	Assigns the current `FitMode`. If `NULL` then the parameter is ignored.
lpnAlignMode	Assigns the current `AlignMode`. If `NULL` then the parameter is ignored.
lpnAspectMode	Assigns the current `AspectMode`. If `NULL` then the parameter is ignored.
lpdAspectValue	Assigns the current `AspectValue`. If `NULL` then the parameter is ignored.

Description:

This function returns the current values of layout parameters. If a parameter is a rectangle then all empty rectangle is returned if it is not set. For a list of possible values see function **`IG_dspl_layout_set()`**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image  */
DWORD  nGrpID;      /* display group identifier */
AT_RECTANGLE ImageRect;    /* Image rectangle  */
AT_RECTANGLE DeviceRect;    /* Device rectangle  */
AT_RECTANGLE ClipRect;      /* Clip rectangle  */
AT_MODE     nFitMode;              /* Fit mode  */
AT_MODE     nAlignMode;           /* Align mode  */
AT_MODE     nAspectMode;          /* Aspect mode  */
DOUBLE      dAspectValue;    /* Aspect value  */

 …
/* get all layout parameters */
```

```
IG_dspl_layout_get( hIGear, nGrpID, &ImageRect, &DeviceRect,
&ClipRect, &nFitMode, &nAlignMode, &nAspectMode, &dAspectValue
);
 …
/* get only device rectangle and fit mode */
IG_dspl_layout_get( hIGear, nGrpID, NULL, &DeviceRect, NULL,
&nFitMode, NULL, NULL, NULL );
 …
```

IG_dspl_layout_set

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_layout_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   DWORD  nFlags,
[IN] const LPAT_RECTANGLE lpImageRect,
[IN] const    LPAT_RECTANGLE lpDeviceRect,
[IN] const LPAT_RECTANGLE lpClipRect,
[IN]   AT_MODE nFitMode,
[IN]   AT_MODE nAlignMode,
[IN]   AT_MODE nAspectMode,
[IN]   DOUBLE  dblAspectValue
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set layout options.
nFlags	Combination of flags where each one specifies if the appropriate parameter is to be used or ignored: `IG_DSPL_IMAGE_RECT` – if `(nFlags&IG_DSPL_IMAGE_RECT)!=0` then the `lpImageRect` parameter of this function is not ignored `IG_DSPL_DEVICE_RECT` – if `(nFlags&IG_DSPL_DEVICE_RECT)!=0` then the `lpDeviceRect` parameter of this function is not ignored `IG_DSPL_CLIP_RECT` – if `(nFlags&IG_DSPL_CLIP_RECT)!=0` then the `lpClipRect` parameter of this function is not ignored `IG_DSPL_FIT_MODE` – if `(nFlags&IG_DSPL_FIT_MODE)!=0` then the `nFitMode` parameter of this function is not ignored `IG_DSPL_ALIGN_MODE` – if `(nFlags&IG_DSPL_ALIGN_MODE)!=0` then the `nAlignMode` parameter of this function is not ignored `IG_DSPL_ASPECT_MODE` – if `(nFlags&IG_DSPL_ASPECT_MODE)!=0` then the `nAspectMode` parameter of this function is not ignored `IG_DSPL_ASPECT_VALUE` – if `(nFlags&IG_DSPL_ASPECT_VALUE)!=0` then the `dblAspectValue` parameter of this function is not ignored
lpImageRect	A new value of `ImageRect` option to set. If `NULL` then it is reset with the default value which means the whole image will be used.
lpDeviceRect	A new value of `DeviceRect` option to set. If `NULL`, then it is reset with the default value (an empty rectangle). This means it is calculated every time from the destination output device.
lpClipRect	A new value of `ClipRect` option to set. If `NULL`, then it is reset with the default value (an empty rectangle). This means it is calculated every time and assigned to the whole client area of the destination output device.
nFitMode	New value of `FitMode` option to set. Possible values are: ■ `IG_DSPL_FIT_TO_DEVICE` ■ `IG_DSPL_FIT_TO_WIDTH` ■ `IG_DSPL_FIT_TO_HEIGHT` ■ `IG_DSPL_ACTUAL_SIZE`

nAlignMode	New value of `AlignMode` option to set. Possible value is OR combination of one constant that sets the horizontal alignment: ■ `IG_DSPL_ALIGN_X_LEFT` ■ `IG_DSPL_ALIGN_X_CENTER` ■ `IG_DSPL_ALIGN_X_RIGHT` and one that sets the vertical alignment: ■ `IG_DSPL_ALIGN_Y_TOP` ■ `IG_DSPL_ALIGN_Y_CENTER` ■ `IG_DSPL_ALIGN_Y_BOTTOM`
nAspectMode	New value of `AspectMode` option to set. Possible values are: ■ `IG_DSPL_ASPECT_FIXED` ■ `IG_DSPL_ASPECT_NOT_FIXED`
dblAspectValue	Sets new value to `AspectValue` option. Possible value may be any positive double.

Description:

This function sets layout parameters. If some of the values are out of range then it does not change and an error is returned.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
DWORD  nGrpID;     /* display group identifier */
AT_RECTANGLE ImageRect;  /* Image rectangle   */
AT_RECTANGLE DeviceRect;  /* Device rectangle   */
AT_RECTANGLE ClipRect;     /* Clip rectangle  */
AT_MODE nFitMode;     /* Fit mode     */
AT_MODE nAlignMode;    /* Align mode     */
AT_MODE nAspectMode;   /* Aspect mode     */
DOUBLE dAspectValue;      /* Aspect value  */
AT_MODE nFlags;        /* flags which specify what to set */

/* set all layout parameters */
nFlags =
IG_DSPL_IMAGE_RECT|IG_DSPL_DEVICE_RECT|IG_DSPL_CLIP_RECT|IG_DSP
L_FIT_MODE|IG_DSPL_ALIGN_MODE|IG_DSPL_ASPECT_MODE|IG_DSPL_ASPEC
T_VALUE;
IG_dspl_layout_get( hIGear, nGrpID, &ImageRect, &DeviceRect,
&ClipRect, nFitMode, nAlignMode, nAspectMode, dAspectValue );

  … …
```

```
/* reset image, device and clip rectangles to its default values
*/
nFlags =
IG_DSPL_IMAGE_RECT|IG_DSPL_DEVICE_RECT|IG_DSPL_CLIP_RECT;
IG_dspl_layout_set( hIGear, nGrpID, nFlags, NULL, NULL, NULL, 0,
0, 0, 0.0 );
 …
/* set new fit mode */
IG_dspl_layout_set( hIGear, nGrpID, IG_DSPL_FIT_MODE, NULL,
NULL, NULL, IG_DSPL_FIT_TO_WIDTH, 0, 0, 0.0 );
 …
/* set align mode so that the image located at right bottom of
device rectangle */
IG_dspl_layout_set( hIGear, nGrpID, IG_DSPL_ALIGN_MODE, NULL,
NULL, NULL, 0, IG_DSPL_ALIGN_X_RIGHT| IG_DSPL_ALIGN_Y_BOTTOM,
0, 0.0 );
 …
```

IG_dspl_LUT_get

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_LUT_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPBYTE lpRLUT,
[OUT]  LPBYTE lpGLUT,
[OUT]  LPBYTE lpBLUT
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to get options.
lpRLUT	Pointer to where the current red lookup table (`RedLut` value) is to be copied. If `NULL`, then this parameter is ignored.
lpGLUT	Pointer where the current green lookup table (`GreenLut` value) is to be copied. If `NULL`, then this parameter is ignored.
lpBLUT	Pointer where the current blue lookup table (`BlueLut` value) is to be copied. If `NULL`, then this parameter is ignored.

Description:

This function returns the current red, green and blue lookup tables `lpRLUT`, `lpGLUT` or `lpBLUT` should be either `NULL` or valid pointers to a 256 byte array.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;          /* HIGEAR handle of image  */
DWORD nGrpID;          /* display group identifier  */
BYTE r_lut[256];        /* red lookup array     */
BYTE g_lut[256];      /* green lookup array     */
 …
IG_dspl_LUT_get( hIGear, nGrpID, r_lut, g_lut, NULL );
 . .
```

IG_dspl_LUT_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_LUT_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nFlags,
[IN]const LPBYTE lpRLUT,
[IN]const LPBYTE lpGLUT,
[IN]const LPBYTE lpBLUT
);
```

Arguments:

hIGear	ImageGear handle of image
dwGrpID	Identifier of group in which to set LUT options.
nFlags	Specify which lookup tables to set. Possible value is 0 or a combination of flags `IG_DSPL_R_CHANNEL` – if this flag is set then the `lpRLUT` parameter of this function is not ignored `IG_DSPL_G_CHANNEL` – if this flag is set then the `lpGLUT` parameter of this function is not ignored `IG_DSPL_B_CHANNEL` – if this flag is set then the `lpBLUT` parameter of this function is not ignored The constant `IG_DSPL_ALL_CHANNELS` is defined for convenience and can be used to set all three channels. `#define IG_DSPL_ALL_CHANNELS (IG_DSPL_R_CHANNEL|IG_DSPL_G_CHANNEL|IG_DSPL_B_CHANNEL)`
lpRLUT	Pointer to a 256 element array of a red lookup table to set. If `NULL`, then identity the array is assigned to `RedLut` option.
lpGLUT	Pointer to a 256 element array of a green lookup table to set. If `NULL`, then identity the array is assigned to `GreenLut` option.
lpBLUT	Pointer to a 256 element array of a blue lookup table to set. If `NULL`, then identity the array is assigned to `BlueLut` option.

Description:

This function assigns new values to red, green and blue lookup tables.

TIP: IG always makes a copy of `lpRLUT`, `lpGLUT` and `lpBLUT` in case they are needed, but does not assign pointers directly so that the application should cleanup the memory allocated for them.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;         /* HIGEAR handle of image  */
DWORD nGrpID;        /* display group identifier  */
BYTE lut[256];      /* lookup array  */
INT i;
 …
/* set inverted lookup table */
for( i = 0; i<256; i++ )
   lut[i] = 255 – i;
```

```
IG_dspl_LUT_set( hIGear, nGrpID, IG_DSPL_ALL_CHANNELS, lut,
lut, lut );
```

IG_dspl_orientation_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_orientation_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPAT_MODE lpnOrientMode
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to retrieve the orientation mode.
lpnOrientMode	Where to copy `OrientMode` value. If `NULL` then this parameter ignored.

Description:

This function returns the current orientation mode. Possible values are listed in the **`IG_dspl_orientation_set()`** function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image  */
DWORD  nGrpID;        /* display group identifier  */
AT_MODE nOrientMode;      /* Orientation mode  */
 …
IG_dspl_orientation_get( hIGear, nGrpID, &nOrientMode );
 …
```

IG_dspl_orientation_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_orientation_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nOrientMode
);
```

Arguments:

hIGear	Image handle where to set orientation mode.
dwGrpID	Identifier of the group in which to set the orientation.

nOrientMode	New `OrientMode` value to set. Possible values are:
	`IG_DSPL_ORIENT_TOP_LEFT`
	`IG_DSPL_ORIENT_LEFT_TOP`
	`IG_DSPL_ORIENT_RIGHT_TOP`
	`IG_DSPL_ORIENT_TOP_RIGHT`
	`IG_DSPL_ORIENT_BOTTOM_RIGHT`
	`IG_DSPL_ORIENT_RIGHT_BOTTOM`
	`IG_DSPL_ORIENT_LEFT_BOTTOM`
	`IG_DSPL_ORIENT_BOTTOM_LEFT`

Description:

This function sets the new orientation mode.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR       hIGear;      /* HIGEAR handle of image  */
DWORD        nGrpID;      /* display group identifier  */

 …
/* rotate image orientation by 90 degree   */
IG_dspl_orientation_set( hIGear, nGrpID,
IG_DSPL_ORIENT_TOP_RIGHT );
 …
```

NOTE: If you are using ImageGear ARTX component in your application please use **IG_dspl_artx_page_print()** function.

IG_dspl_palette_create

Group: Display

Declaration:

```
AT_ERRCOUNT     ACCUAPI   IG_dspl_palette_create(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[OUT]  HPALETTE FAR* lphPalette
);
```

Arguments:

hIGear	ImageGear handle of image.
DwGrpID	Identifier of group containing the palette options to use.
HWnd	Handle of window where the image is drawn.
HDC	Handle of device context on which to draw.
LphPalette	Pointer to where the palette handle should be returned.

Description:

This function calculates all the display parameters, the way **IG_dspl_image_draw()** does, and produces the same palette that `IG_dspl_image_draw` realizes before drawing onto `hDC`. It returns `DevicePalette` if it is not set with a custom value.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image  */
DWORD  nGrpID;      /* display group identifier */
HPALETTE hPalette;     /* handle of palette  */
 …
IG_dspl_palette_create( hIGear, nGrpID, hWnd, hDC, &hPalette );
```

...

IG_dspl_palette_handle

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_palette_handle(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   HDC    hDC,
[IN]   AT_MODE nPalMode,
[OUT]  LPBOOL lpbRealized
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group that should be used.
hWnd	Valid handle of window where image is drawn and from which palette message is to be handled.
hDC	Handle of device context where image is to be drawn. If NULL, then it will automatically be retrieved from hWnd.
nPalMode	New value of PaletteMode to set. Possible value is either IG_DSPL_PALETTE_HIGH or IG_DSPL_PALETTE_LOW. Please note that this new value is assigned only if the current value of PaletteMode is not IG_DSPL_PALETTE_DISABLE. If the current value is IG_DSPL_PALETTE_DISABLE then the function does not perform any action (to change this, use **IG_dspl_palette_set()** function).
lpbRealized	Pointer to the BOOL parameter where TRUE will be assigned if new palette has been realized and FALSE if otherwise. If NULL, then this parameter is ignored.

Description:

This function is designed to be used to handle palette messages from the operating system. Handling of those messages is extremely important for palette based devices.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image   */
DWORD  nGrpID;      /* display group identifier */

 …
case WM_QUERYNEWPALETTE:
   /* let Image Gear handle palette management */
IG_dspl_palette_handle(hIGear, nGrpID, hWnd, NULL,
IG_DSPL_PALETTE_HIGH, &bRealized );
   return bRealized;

case WM_PALETTECHANGED:
   /* let Image Gear handle palette management */
IG_dspl_palette_handle(hIGear, nGrpID, hWnd, NULL,
IG_DSPL_PALETTE_LOW, NULL );
   break;
 …
```

IG_dspl_palette_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_palette_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPAT_MODE lpnPalMode,
[OUT]  HPALETTE FAR* lphPalette
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to get palette options.
lpnPalMode	Pointer to where `PaletteMode` is to be received. If `NULL`, then this parameter is ignored.
lphPalette	Pointer to where `DevicePalette` is to be received. If `NULL`, then this parameter is ignored.

Description:

This function returns the current palette settings.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image   */
DWORD  nGrpID;      /* display group identifier  */
AT_MODE nPalMode;    /* palette mode  */
HPALETTE hPalette;    /* palette handle */

 …
IG_dspl_palette_get( hIGear, nGrpID, &nPalMode, &hPalette );
 …
```

IG_dspl_palette_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_palette_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nPalMode,
[IN]   HPALETTE hPalette
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set palette options.
nPalMode	New value of `PaletteMode` to set. Possible values are: ■ `IG_DSPL_PALETTE_HIGH` ■ `IG_DSPL_PALETTE_LOW` ■ `IG_DSPL_PALETTE_DISABLE`
hPalette	New value of `DevicePalette` option. Possible value is `NULL` or valid OS-dependent palette handle. Please note that the application code is responsible for removing all the resources allocated for `DevicePalette` if it is not `NULL`.

Description:

This function sets the new palette options.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
DWORD  nGrpID;    /* display group identifier  */
 …
/* disable palette realization */
IG_dspl_palette_set( hIGear, nGrpID, IG_DSPL_PALETTE_DISABLE,
NULL );
 …
```

IG_dspl_PPM_correct_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_PPM_correct_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPBOOL lpbEnable
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in from which to get options.
LpbEnable	Pointer to where to get the current value of option `PPMCorrect`.

Description:

This function returns the current value of options which specify either to use the image resolution or the image dimension to calculate the aspect ratio.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image  */
DWORD  nGrpID;      /* display group identifier    */
BOOL   bPPMEnable;    /* PPM corect flag value  */
```

```
 …
IG_dspl_PPM_correct_get ( hIGear, nGrpID, &bPPMEnable );
 …
```

IG_dspl_PPM_correct_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_PPM_correct_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   BOOL   bEnable
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group for which to set options.
bEnable	New value for PPMCorrect option. If TRUE then the resolution will be used to calculate the aspect ratio, but if FALSE then the image dimension will be used.

Description:

This function sets the new value of the PPMCorrect option.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image   */
DWORD  nGrpID;     /* display group identifier */
 …
IG_dspl_PPM_correct_set ( hIGear, nGrpID, TRUE );
 …
```

IG_dspl_resize_handle

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_resize_handle(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd );
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group which is used to display the image on the hWnd.
hWnd	Handle of window where the image is displayed.

Description:

This function is to be called during WM_SIZE message handling and it recalculates all the parameters according to the new window width and height.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image  */
DWORD  nGrpID;        /* display group identifier */

 …
case WM_SIZE:
   /* handle window resizing */
   IG_dspl_resize_handle( hIGear, nGrpID, hWnd );
   break;
 …
```

IG_dspl_ROP_get

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_ROP_get(
HIGEAR    hIGear,
DWORD     dwGrpID,
```

```
LPDWORD    lpdwROP
);
```

Arguments:

hIGear	HIGEAR handle of image
dwGrpID	Identifier of display group to get ROP code from
lpdwROP	Returned ROP code

Description:

This function gets the ROP (raster-operation) code used to display images on Windows platforms. This code determines how the source image pixels are combined with the destination area pixels. The default ROP is SRCCOPY, which overwrites the destination area with the source image. Other codes are described in the documentation for the Windows GDI function BitBlt.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
HIGEAR       hIGear;      /* HIGEAR handle of image */
DWORD        nGrpID;      /* display group identifier */
DWORD        dwROP;       /* retrieved ROP code */
 ...
/* get ROP code */
IG_dspl_ROP_get( hIGear, nGrpID, &dwROP );
 ...
```

IG_dspl_ROP_set

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_ROP_set(
HIGEAR hIGear,
DWORD dwGrpID,
DWORD dwROP
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set ROP code.
dwROP	ROP code to set.

Description:

This function sets the ROP (raster-operation) code to use when displaying images on Windows platforms. This code determines how the source image pixels are combined with the destination area pixels. The default ROP is SRCCOPY, which overwrites the destination area with the source image. Other codes are described in the documentation for the Windows GDI function BitBlt.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR       hIGear;      /* HIGEAR handle of image  */
DWORD        nGrpID;      /* display group identifier  */
 ...
/* image will be XOR'd with destination contents */
IG_dspl_ROP_set( hIGear, nGrpID, SRCINVERT );
 ...
```

IG_dspl_scroll_get

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_scroll_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[OUT]  LPAT_MODE lpnScrollMode,
[OUT]  LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to retrieve scroll parameters.
hWnd	Handle of window where image is drawn.
lpnScrollMode	Where `ScrollbarMode` is to be received. If `NULL`, then this parameter is ignored.
lpScrollInfo	Where to copy the current scroll parameters. If `NULL`, then this parameter is ignored.

Description:

This function calculates and returns the current scroll parameters.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image   */
DWORD  nGrpID;        /* display group identifier  */
AT_MODE nScrollMode;      /* scroll mode   */
AT_SCROLL_INFO  ScrollInfo;   /* scroll info  */

 …
IG_dspl_scroll_get( hIGear, nGrpID, hWnd, &nScrollMode,
&ScrollInfo );
 …
```

IG_dspl_scroll_handle

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_scroll_handle(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   AT_MODE nScrlType,
[IN]   AT_MODE nScrlMode,
[IN]   LONG   lScrlValue,
[OUT]  LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group which will be used to perform operation.
hWnd	Handle of window where image is displayed and where scrolling is performed.
nScrlType	Scroll command. Its value is platform dependent. For Win32 platforms there should be LOWORD(wParam) and for Win16, wParam. Valid values are: ■ IG_DSPL_SCROLL_HORIZONTAL ■ IG_DSPL_SCROLL_VERTICAL
nScrlMode	Scroll command. Its value is platform dependent. For Win32 platforms there should be LOWORD(wParam) and for Win16, wParam. Valid values are: ■ IG_DSPL_HSCROLLBAR ■ IG_DSPL_HSCROLLBAR_AUTO ■ IG_DSPL_HSCROLLBAR_ENABLE ■ IG_DSPL_HSCROLLBAR_DISABLE ■ IG_DSPL_VSCROLLBAR ■ IG_DSPL_VSCROLLBAR_AUTO ■ IG_DSPL_VSCROLLBAR_ENABLE ■ IG_DSPL_VSCROLLBAR_DISABLE
nScrlValue	Scroll value. It is also platform dependent and its value may have a different meaning from nScrlType. Under the Windows platform, it is not used because the scroll position is obtained directly from the scrollbars.
lpScrollInfo	Receive updated scroll parameters (not scrollbar). If NULL then it is ignored.

Description:

This function is designed to handle scrollbar messages from an hWnd window. It should not be used to set the absolute scroll position; function **IG_dspl_scroll_to()** should be used instead.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image      */
DWORD  nGrpID;       /* display group identifier     */
 …
case WM_HSCROLL:           /* horizontal scrolling   */
```

```
IG_dspl_scroll_handle( hIGear, nGrpID, hWnd,
IG_DSPL_SCROLL_HORIZONTAL, LOWORD(wParam), HIWORD(wParam), NULL
);
   break;

case WM_VSCROLL: /* vertical scrolling */
IG_dspl_scroll_handle( hIGear, nGrpID, hWnd,
IG_DSPL_SCROLL_VERTICAL, LOWORD(wParam), HIWORD(wParam), NULL
);
   break;
 …
```

IG_dspl_scroll_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_scroll_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   AT_MODE nScrollMode,
[IN]   INT    nXPage,
[IN]   INT    nYPage,
[OUT]  LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set scroll options.
hWnd	Handle of window where image is drawn.
nScrollMode	New value of `ScrollbarMode` to set. Possible value is a combination of the horizontal scrollbar flag: ■ `IG_DSPL_HSCROLLBAR_AUTO` ■ `IG_DSPL_HSCROLLBAR_ENABLE` ■ `IG_DSPL_HSCROLLBAR_DISABLE` and the vertical: ■ `IG_DSPL_VSCROLLBAR_AUTO` ■ `IG_DSPL_VSCROLLBAR_ENABLE` ■ `IG_DSPL_VSCROLLBAR_DISABLE`
nXPage	New value of scrolling page width. If 0 then it will be calculated from `ClipRect`.
nYPage	New value of scrolling page height. If 0 then it will be calculated from `ClipRect`.
lpScrollInfo	Where to copy new scroll parameters. If `NULL` then the parameter is ignored.

Description:

This function sets the scroll parameters (not scroll position) and allows you to enable and disable the vertical and horizontal scrollbars associated with a given window.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image  */
DWORD  nGrpID;        /* display group identifier   */
AT_SCROLL_INFO  ScrollInfo;      /* scroll info  */

 …
/* always hide both scrollbars */
IG_dspl_scroll_set( hIGear, nGrpID, hWnd,
IG_DSPL_HSCROLLBAR_DISABLE|IG_DSPL_VSCROLLBAR_DISABLE, 0, 0,
&ScrollInfo );
 …
```

IG_dspl_scroll_to

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_scroll_to(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   INT    nXPos,
[IN]   INT    nYPos,
[OUT]  LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group where to work.
hWnd	Handle of window where image is drawn and scrolled.
nXPos	New horizontal scroll position. Should be in a valid range that can be retrieved using the **IG_dspl_scroll_get()** function.
nYPos	New vertical scroll position. Should be in a valid range that can be retrieved using the **IG_dspl_scroll_get()** function.
lpScrollInfo	Where to copy the new scroll parameters. If NULL then the parameter is ignored.

Description:

This function scrolls the image to a specified position and updates the window's scroll bars accordingly. It can be used even if a window's scroll bars are disabled. Both scroll positions should be in valid scroll range but if not, then the nearest valid value will be assigned.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;         /* HIGEAR handle of image   */
DWORD  nGrpID;          /* display group identifier  */
AT_SCROLL_INFO  ScrollInfo;    /* scroll info  */


 …
/* scroll down and right from current position */
```

```
IG_dspl_scroll_get( hIGear, nGrpID, hWnd, NULL, &ScrollInfo );
If( (ScrollInfo.h_cur_pos<ScrollInfo.h_max) &&
(ScrollInfo.v_cur_pos<ScrollInfo.v_max) )
   IG_dspl_scroll_to( hIGear, nGrpID, hWnd,
ScrollInfo.h_cur_pos+1, ScrollInfo.v_cur_pos+1, NULL );
```

IG_dspl_scroll_to_ex

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_scroll_to_ex(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN]   INT    nXPos,
[IN]   INT    nYPos,
[OUT]  LPAT_SCROLL_INFOlpScrollInfo
[IN]   BOOL   bRepaint
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group where to work.
hWnd	Handle of window where image is drawn and scrolled.
nXPos	New horizontal scroll position. Should be in a valid range that can be retrieved using the **IG_dspl_scroll_get()** function.
nYPos	New vertical scroll position. Should be in a valid range that can be retrieved using the **IG_dspl_scroll_get()**function.
lpScrollInfo	Where to copy the new scroll parameters. If NULL then the parameter is ignored.
bRepaint	This parameter determines if the function repaints scrolling content (TRUE) or not (FALSE).

Description:

This function scrolls the image to a specified position and updates the window's scroll bars accordingly. It can be used even if a window's scroll bars are disabled. Both scroll positions should be in valid scroll range but if not, then the nearest valid value will be assigned.

This function is the same as **IG_dspl_scroll_to()** function, but has bRepaint parameter that when FALSE allows do not repaint the scrolling content.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_dspl_transparency_get

Group: Display

Declaration:

```
AT_ERRCOUNT    ACCUAPI  IG_dspl_transparency_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[OUT]  LPAT_MODE lpnTranspMode,
[OUT]  LPAT_RGB lpTranspColor,
[OUT]  LPHIGEAR lphIMask,
[OUT]  LPAT_RECTANGLE lpMaskRect,
[OUT]  LPAT_POINT lpMaskLocation
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group to use.
lpnTranspMode	Pointer where `TranspMode` options are returned. If `NULL`, then this parameter is ignored.
lpTranspColor	Pointer where `TranspColor` option is returned. If `NULL`, then this parameter is ignored.
hphIMask	Pointer where `TranspMask` option is returned. If `NULL`, then this parameter is ignored. Please note that this function does not change or delete the value of `TranspMask`, but only makes its copy.
lpMaskRect	Pointer to where to return the value of the `MaskRect` option. If `NULL`, then this parameter is ignored. If an empty rectangle is returned then it is initialized with a rectangle equal to the entire mask image.
lpMaskLocation	Pointer to where to return the value of `MaskLocation` option. If `NULL`, then this parameter is ignored.

Description:

This function returns the transparency parameters.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image    */
DWORD  nGrpID;         /* display group identifier */
AT_MODE nTranspMode;     /* transparency mode  */
AT_RGB TranspColor;    /* transparency color  */
HIGEAR hIMask;         /* transparent mask    */
AT_RECTANGLE MaskRect;       /* mask rectangle      */
AT_POINT MaskLocation;    /* mask location   */

 …
IG_dspl_transparency_get( hIGear, nGrpID, &nTranspMode,
&TranspColor, &hIMask, &MaskRect, &MaskLocation );

if( IG_image_is_valid( hIMask ) )
   IG_image_delete( hIMask );
 …
```

IG_dspl_transparency_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_transparency_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nTranspMode,
[IN] constLPAT_RGB lpTranspColor,
[IN]   HIGEAR hIMask,
[IN] const LPAT_RECTANGLE lpMaskRect,
[IN] const LPAT_POINT lpMaskLocation
);
```

Arguments:

hIGear	ImageGear handle of image
dwGrpID	Identifier of the group in which to set transparency options.

nTranspMode	New value of `TranspMode` to set. Possible value maybe: ■ `IG_DSPL_TRANSPARENCY_NONE,` or a combination of one of the following flags: ■ `IG_DSPL_TRANSPARENCY_COLOR` ■ `IG_DSPL_TRANSPARENCY_MASK` ■ `IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE` and one of these flags: ■ `IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE` ■ `IG_DSPL_TRANSPMASK_LOCATE_TO_CLIPRECT` ■ `IG_DSPL_TRANSPMASK_LOCATE_ABSOLUTE`
lpTranspColor	New value of `TranspColor` to set. If `NULL`, then this parameter is ignored.
hIMask	New value of `TranspMask` option to set. Please note that the image is not deleted and the application code is responsible for removing the old value of this option.
lpMaskRect	New value of `MaskRect` option to set. If `NULL`, then this parameter is ignored. An empty rectangle is possible.
lpMaskLocation	New value of `MaskLocation` option to set. If `NULL`, then this parameter is ignored.

Description:

This function sets the transparency options.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image  */
DWORD  nGrpID;        /* display group identifier  */
AT_MODE nTranspMode;    /* transparency mode  */
AT_RGB TranspColor;    /* transparency color */
HIGEAR hIMask;         /* transparent mask   */
 …
/* enable transparent color and mask with default mask rectangle
and location oriented and scaled with main image */
nTranspMode =
IG_DSPL_TRANSPARENCY_COLOR|IG_DSPL_TRANSPARENCY_MASK|IG_DSPL_TR
ANSPMASK_STRETCH_TO_IMAGE|IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE;
/* set transparent color to white */
TranspColor.r = TranspColor.g = TranspColor.b = 255;
IG_dspl_transparency_set( hIGear, nGrpID, nTranspMode,
&TranspColor, hIMask, NULL, NULL );
 …
```

IG_dspl_update_rect_get

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_update_rect_get(
[IN]   HIGEARhIGear,
[IN]   DWORDdwGrpID,
[OUT]LPAT_RECTANGLElpUpdateRect
);
```

Arguments:

hIGear	Image handle from which to search the rectangle.
dwGrpID	Identifier of the group from which to get options.
lpUpdateRect	Pointer to rectangle in which to copy the value of **UpdateRect** option. If **NULL** then it is ignored. If **UpdateRect** is not set then an empty rectangle is returned.

Description:

This function returns the current values of update rectangle. If a parameter is a rectangle, and this parameter is not set, then an empty rectangle is returned.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Samples: filters

See also:

Geometric Layout

IG_dspl_update_rect_set

Group: Display

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_update_rect_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN] constLPAT_RECTANGLElpUpdateRect
);
```

Arguments:

hIGear	ImageGear handle of an image.
dwGrpID	Identifier of the group in which to set update rectangle.
lpUpdateRect	Sets new value to **UpdateRect** option.

Description:

This function sets the new value to update rectangle. Usually this value is received from Expose X event.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
void  wndDrawImage_callback( Widget w, XtPointer data, XtPointer
call_data )
{
   XmDrawingAreaCallbackStruct*cbs =
             (XmDrawingAreaCallbackStruct*)call_data;
   XExposeEvent*event = (XExposeEvent*)cbs->event;
   AT_RECTANGLEUpdateRect;

   /* Redisplay only on last event of the series */
   if( IG_image_is_valid( hIGear ) )
   {
      /* Set the dirty rectangle for repaint */
      UpdateRect.x= event->x;
      UpdateRect.y= event->y;
      UpdateRect.width= event->width;
      UpdateRect.height = event->height;
      IG_dspl_update_rect_set( hIGear, IG_GRP_DEFAULT,
&UpdateRect );
      /* Repaint window on expose events */
      IG_dspl_image_draw( hIGear, IG_GRP_DEFAULT, XtWindow(w),
XDefaultGCOfScreen(XtScreen(w)), NULL );
   }
}
```

Samples: Filters

See also:

Geometric Layout

IG_dspl_zoom_get

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_zoom_get(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[OUT]  LPAT_MODE lpnZoomMode,
[OUT]  LPDOUBLE lpdblHZoom,
[OUT]  LPDOUBLE lpdblVZoom
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of the group from which to obtain zoom parameters.
hWnd	Handle of window where the image is drawn.
lpnZoomMode	Pointer to where `ZoomMode` is to be received. If `NULL`, then this parameter is ignored.
lpdblHZoom	Where to return the calculated horizontal zoom value. This value is always calculated whether or not the horizontal zoom is fixed.
lpdblVZoom	Where to return the calculated vertical zoom value. This value is always calculated whether or not the vertical zoom is fixed.

Description:

This function returns the current zoom values.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image  */
DWORD  nGrpID;        /* display group identifier  */
AT_MODE nZoomMode;     /* zoom mode     */
DOUBLE dHZoom;        /* horizontal zoom value   */
DOUBLE dVZoom;        /* vertical zoom value    */

 …
IG_dspl_zoom_get( hIGear, nGrpID, hWnd, &nZoomMode, &dHZoom,
&dVZoom );
```

...

IG_dspl_zoom_set

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_zoom_set(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   AT_MODE nZoomMode,
[IN]   DOUBLE dHZoom,
[IN]   DOUBLE dVZoom
);
```

Arguments:

hIGear	Image handle to where set zoom options.
dwGrpID	Identifier of group in which to set zoom options.
nZoomlMode	New value of `ZoomMode` to set. Possible value is combination from one of the horizontal zoom flags: ■ `IG_DSPL_ZOOM_H_NOT_FIXED` ■ `IG_DSPL_ZOOM_H_FIXED` and one of the vertical flags: ■ `IG_DSPL_ZOOM_V_NOT_FIXED` ■ `IG_DSPL_ZOOM_V_FIXED`
dHZoom	New value of `ZoomValueH` to set. Any positive number is accepted. Value 1.0 means that the image is zoomed by 100% or displayed in its actual size.
dVZoom	New value of `ZoomValueV` to set. Any positive number is accepted. Value 1.0 means that the image is zoomed by 100% or displayed in its actual size.

Description:

This function sets the new zoom values.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image    */
DWORD  nGrpID;        /* display group identifier  */
```

...

```
/* set aspect ratio as not fixed and zoom image by 150% and 200%
for horizontal and vertical directions respectively  */
IG_dspl_layout_set( hIGear, nGrpID, IG_ASPECT_MODE, NULL, NULL,
NULL, 0, 0, IG_DSPL_ASPECT_NOT_FIXED, 0.0 );
IG_dspl_zoom_set( hIGear, nGrpID, IG_DSPL_ZOOM_H_FIXED|
IG_DSPL_ZOOM_V_FIXED, 1.5, 2.0 );
 …
/* the same as before but only for horizontal direction;
vertical direction is zoomed according to current fit method and
device rectangle */
IG_dspl_zoom_set( hIGear, nGrpID, IG_DSPL_ZOOM_H_FIXED|
IG_DSPL_ZOOM_V_NOT_FIXED, 1.5, 2.0 );
 …
```

IG_dspl_zoom_to_rect

Group: Display

Declaration:

```
AT_ERRCOUNT   ACCUAPI   IG_dspl_zoom_to_rect(
[IN]   HIGEAR hIGear,
[IN]   DWORD  dwGrpID,
[IN]   HWND   hWnd,
[IN] const LPAT_RECTANGLE lpZoomRect,
[OUT]  LPDOUBLE lpdblHZoom,
[OUT]  LPDOUBLE lpdblVZoom
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group in which to set zoom_to_rect options.
hWnd	Handle of window where image is drawn.
lpZoomRect	Pointer to the rectangle in device coordinates that should be fitted into `ClipRect`.
lpdblHZoom	Pointer to where to return the newly calculated `ZoomValueH` value. If `NULL`, then this parameter is ignored.
lpdblVZoom	Pointer to where to return the newly calculated `ZoomValueV` value. If `NULL`, then this parameter is ignored.

Description:

This function calculates the zoom and scroll values, so that the specified rectangle is fitted to `ClipRect`. It assigns `ZoomMode=IG_DSPL_ZOOM_H_FIXED|IG_DSPL_ZOOM_V_FIXED`, calculates and modifies `ZoomValueH`, `ZoomValueV`, and the horizontal and vertical scroll positions. It does not change the value of rectangles `DeviceRect`, `ImageRect` and `ClipRect`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image  */
DWORD  nGrpID;        /* display group identifier */
AT_RECTANGLE ZoomRect;     /* rectangle where to zoom  */
RECT   rc;

 …
GetClientRect( hWnd, &rc );
ZoomRect.x = rc.left;
ZoomRect.y = rc.top;
ZoomRect.width = (rc.right – rc.left + 1)/2;
ZoomRect.height = (rc.bottom – rc.top + 1)/2;

IG_dspl_zoom_to_rect( hIGear, nGrpID, hWnd,  &ZoomRect, NULL,
NULL );
 …
```

IG_err_callback_get

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_callback_get(
[IN]   LPVOID FAR*   lplpPrivate,
[IN]   LPFNIG_ERRSTACK_ADD FAR*lplpfnAddCB,
[IN]   LPFNIG_ERRSTACK_CLEAR FAR*lplpfnClearCB
);
```

Arguments:

lplpPrivate	Pointer indicating where to receive private data.
lplpfnAddCB	Pointer indicating where to receive the pointer to the callback function that is called after the record is added to the error stack.
lplpfnClearCB	Pointer indicating where to receive the pointer to the callback function that is called after the error stack is cleared.

Description:

This function is currently associated with the error stack callback data and functions that are called to flag error stack changes.

NOTE: See also **IG_err_callback_set()** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the error code that executes this function. A value of zero means that no errors have occurred.

Example:

```
LPVOID                lpPrivate;
LPFNIG_ERRSTACK_ADD   lpfnAdd;
LPFNIG_ERRSTACK_CLEAR lpfnClear;
...
```

IG_err_callback_set

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_callback_set(
[IN]LPVOID   lpPrivate,
[IN]LPFNIG_ERRSTACK_ADD lpfnAddCB,
[IN]LPFNIG_ERRSTACK_CLEAR lpfnClearCB
);
```

Arguments:

lpPrivate	Any private data. NULL is acceptable.
lpfnAddCB	Pointer to the callback function that will be called after the record is added to the error stack. NULL is acceptable.
lpfnClearCB	Pointer to the callback function that will be called after the error stack is cleared. NULL is acceptable.

Description:

This is the associated callback data, which functions with the error stack that flags error stack changes. Declare the following types for this function:

```
typedef VOID (LPACCUAPI LPFNIG_ERRSTACK_ADD)(
LPVOID lpPrivate, - pointer to private data
UINT   nRecord, - index of error record in the stack
INT    iLineNumber, - the number of line where error occured
AT_ERRCODE iCode, - error code
UINT   nLevel, - level of error, 0 - critical error (stop
execution), >0 - warning of different levels
```

```
LONG   lValue1, - associated long value
LONG   lValue2, - associated long value
LPCHAR lpFileName, - string of source file name where error
occured
LPCHAR lpExtratext - additional error description
);

typedef VOID (LPACCUAPI LPFNIG_ERRSTACK_CLEAR)(
LPVOID lpPrivate, -  pointer to private data.
UINT   nRecords - number of records cleared from the stack
starting from 0 index.
);
```

NOTE: See also **`IG_err_callback_get()`** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the error code that executes this function. A value of zero means no errors have occurred.

Example:

```
 …
/* set callback function to output error records to the window
with handle hWnd */
IG_err_callback_set( (LPVOID)hWnd, ErrAdd, ErrClear );
 …
}
VOID ACCUAPI ErrAdd(
LPVOID lpPrivate,
UINT   nRecord,
INT    iLineNumber,
AT_ERRCODE iCode,
UINT   nLevel,
LONG   lValue1,
LONG   lValue2,
LPCHAR lpFileName,
LPCHAR lpExtratext
)
{
   HWND   hWnd = (HWND)lpPrivate;
  …
   /* update error window with new records */
}

VOID ACCUAPI ErrClear(
LPVOID lpPrivate,
UINT   nRecords
```

```
)
{
   HWND   hWnd = (HWND)lpPrivate;
  …
   /* remove records from error window */
}
```

IG_err_count_get

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_count_get( VOID );
```

Arguments:

(none)

Description:

This function returns the total number of records (errors + warnings) in the error stack.

NOTE: See also **IG_err_callback_get()** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the total count of records in the error stack.

Example:

```
AT_ERRCOUNT iErrCount;

IErrCount = IG_err_count_get( );
```

IG_err_error_check

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_error_check(
[IN] UINT nLevel
);
```

Arguments:

nLevel	Level of errors to count.

Description:

This function counts number of records of the specified level in the error stack.

NOTE: See also **IG_err_callback_get()** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the number of records in the stack of the given level.

Example:

```
AT_ERRCOUNT iErrCount;

/* get number of records of level 0 "
iErrCount = IG_err_error_check( 0  );
```

IG_err_error_get

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_error_get(
[IN]   UINT   nLevel,
[IN]   UINT   nIndex,
[IN]   LPCHAR lpsFileName,
[IN]   UINT   nFNameSize,
[IN]   LPINT  pnLineNumber,
[IN]   LPAT_ERRCODE lpnCode,
[IN]   LPAT_INT lplValue1,
[IN]   LPAT_INT lplValue2,
[IN]   LPCHAR lpExtraText,
[IN]   UINT   nETextSize
);
```

Arguments:

nLevel	Level of errors to to index.
Nindex	Zero-based record index of the given level
lpsFileName	Pointer indicating where to return the source's file name where the error occurred.
nFNameSize	Size of the memory buffer `lpsFileName`.
lpnLineNumber	Pointer indicating where to return the line number where the error occurred.
LpnCode	Pointer indicating where to return the error code.
lplValue1	Pointer indicating where to return the first associated long value.
lplValue2	Pointer indicating where to return the second associated long value.
lpExtraText	Pointer indicating where to return the additional text description.
nETextSize	The size of the memory `lpExtraText`.

Description:

This function returns information about the record from error stack with the given index. The difference between this function and **`IG_err_record_get()`** is that this index exists for records with the specified level, `nLevel`; not for all records of in the stack.

NOTE: See also **`IG_err_callback_get()`** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the error code when this function is executed. A value of zero means no errors have occurred.

Example:

```
AT_ERRCOUNT iErrCount, i;
CHAR         FileName[_MAX_PATH];
INT          nLineNumber;
AT_ERRCODE   nCode;

/* get all errors of level 0 */
iErrCount = IG_err_error_check( 0 );
for( i = 0; i<iErrCount; i++ )
{
   IG_err_error_get( 0, i, FileName, sizeof(FileName),
&nLineNumber, &nCode, NULL, NULL, NULL, 0 );
```

```
    …
}
```

IG_err_error_set

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_error_set(
[IN]const LPCHAR lpFileName,
[IN]INT   iLineNumber,
[IN]AT_ERRCODE nCode,
[IN]UINT  nLevel,
[IN]AT_INT lplValue1,
[IN]AT_INT lplValue2,
[IN]constLPCHAR lpExtraText,
);
```

Arguments:

lpFileName	A long pointer to a string that supplies the name of the module from which the error was generated. It is recommended that you use the `FILE` constant in this field.
iLineNumber	An integer number indicating the line from which the error was set. It is recommended that you use the `_LINE_` constant in this field.
nCode	An integer value of type `AT_ERRCODE`. Set this to the code number of the error that you wish to place on the error stack. If you will be placing an error on the stack that you have defined yourself, be sure to set it to a number that is less than the current value of `IG_LAST_ERROR_NUMBER`.
nLevel	The level of error. 0 means most critical error (stop execution). 1,2,… means records of type "warning."
lplValue1 lplValue2	These LONG arguments are available so that you may supply any supporting information about the error. Your application might use these values to decide what to do after setting a particular kind of error.
lpExtraText	Additional text description of the error. It can be `NULL` if it is not available.

Description:

This function is called to place an ImageGear error on the error stack.

Bits Per Pixel: Not applicable

Return Value:

Returns the error code when this function is executed. A value of zero means no errors have occurred.

Example:

```
/* set application specific waining */
IG_err_error_set( FILE, LINE, -1, 2, 0, 0, "Some explanation"
);
```

IG_err_record_get

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_record_get(
[IN]UINT  nIndex,
[IN]LPCHAR lpsFileName,
[IN]UINT  nFNameSize,
[IN]LPINT lpnLineNumber,
[IN]LPAT_ERRCODE lpnCode,
[IN]LPUINT lpnLevel,
[IN]LPAT_INT lplValue1,
[IN]LPAT_INT lplValue2,
[IN]LPCHAR lpExtraText,
[IN]UINT  nETextSize
);
```

Arguments:

nIndex	Zero based index of the record.
lpsFileName	Pointer indicating where to return the source's file name where the error occurred.
nFNameSize	The buffer memory size of `lpsFileName`.
lpnLineNumber	Pointer indicating where to return the line number where the error occurred.
lpnCode	Pointer indicating where to return the error code.
lpnLevel	Pointer indicating where to return the error level.
lplValue1	Pointer error where to return the first associated long value.
lplValue2	Pointer where to return the second associated long value.
lpExtraText	Pointer indicating where to return additional text description.
nETextSize	The size of memory `lpExtraText`.

Description:

This function returns information about the record from the error stack with the given index. This index is the general index of all records in the stack. The difference between this function and **`IG_err_error_get()`** is that this function enumerates all records rather than only records of a given level.

NOTE: See also **`IG_err_callback_get()`** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the error code of executing of this function. A value of zero means no errors have occurred.

Example:

```
AT_ERRCOUNT iErrCount, i;
CHAR         FileName[_MAX_PATH];
INT          nLineNumber;
AT_ERRCODE   nCode;
UINT         nLevel;

/* get all records from error stack. */
iErrCount = IG_err_count_get( );
for( i = 0; i<iErrCount; i++ )
{
```

```
   IG_err_record_get( i, FileName, sizeof(FileName),
&nLineNumber, &nCode, &nLevel, NULL, NULL, NULL, 0 );
  …
}
```

IG_err_stack_clear

Group: Library Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_stack_clear( VOID );
```

Arguments:

None.

Description:

This function remove all records from the error stack.

NOTE: See also **IG_err_callback_get()** function.

Bits Per Pixel: Not applicable

Return Value:

Returns the error code of executing this function. A value of zero means no errors have occurred.

Example:

```
/* clear error stack */
IG_err_error_clear();
```

IG_error_check

Group: Library Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_error_check( VOID );
```

Arguments:

None.

Description:

Returns to you the number of errors currently on the ImageGear error stack. A value of zero means no ImageGear errors have occurred during your last `IG_` function call.

NOTE: See also **`IG_error_get()`**, **`IG_error_set()`**, and **`IG_error_clear()`** functions.

Bits Per Pixel: Not applicable

Return Value:

This function returns the number of errors currently on the ImageGear error stack. Since each `IG_` function, excluding all `IG_dspl_...()` functions, cleares the error stack automatically upon entry, this will equal the number of errors that occurred during your last `IG_` function call.

Example:

See the example for function **`IG_error_clear()`**.

IG_error_clear

Group: Library Utility

Declaration:

```
void ACCUAPI IG_error_clear( VOID );
```

Arguments: None

Description:

Clears all errors from the error stack. After calling this function, **`IG_error_check()`** will return zero.

Bits Per Pixel: Not applicable

Example:

```
BYTE szModuleName[30]; /* Will hold ret'd module name, up to 29
chars */
INT  iNameArraySize; /* Will hold size of that BYTE array */
INT  iLineNumber;      /* Will hold returned line number */
AT_ERRCODE iCode;   /* Will hold returned error code */
AT_ERRCOUNT nErrcount; /* Will hold count of errors on stack */
INT n;       /* Will be Loop Index and Error Index Number */
iNameSize = 30; /* Size of module-name array */
nErrcount = IG_error_check();/* Obtain number of errors on stack
*/
```

```
for ( n = 0; n < nErrcount; n++ )/* For each error on stack */
   {
/* Get Error Code, Module Name, Line Number:  */
   IG_error_get( n, &szModuleName, iNameSize,
&iLineNumber, &iCode, NULL, NULL );
   /* … Process or Display Error, etc. */
   }
IG_error_clear ();/* Done getting errs, clear error stack */
```

IG_error_get

Group: Library Utility

Declaration:

```
void ACCUAPI IG_error_get (INT iErrorIndex,
   LPSTR  szFileName,
   INT    cbFileNameSize,
   LPINT  lpiLineNumber,
   LPAT_ERRCODE lpiCode,
   );
```

Arguments:

iErrorIndex	A value of 0 means fetch the first error placed on the stack. Tells which error to fetch from stack.
szFileName	Far pointer to where to return the module name in which this error occurred. If this pointer is `NULL`, the module name is not returned
cbFileNameSize	Number of bytes available in byte array pointed to by `szFileName`
lpiLineNumber	Far pointer to where to return the line number at which the error occurred. If `NULL`, the line number is not returned
lpiCode	Far pointer to where to return the Error Code. If `NULL`, the Error Code is not returned

Description:

This function is called to fetch an ImageGear Error Code and associated information from the error stack.

Set `iErrorIndex` to indicate which error you wish to fetch. `iErrorIndex = 0` means the error added to the stack first. The other arguments (except `cbFileNameSize`) are `FAR` pointers telling this function where to return the retrieved information to you. This information consists of the Error Code, the module name and line number at which the error occurred. See the "Appendix A: Function Error Return Codes" for a list of all ImageGear Error Codes .

NOTE: To determine the number of errors currently on the error stack use **IG_error_check()**. After fetching all error information you need using IG_error_get(), use **IG_error_clear()** to clear the stack. See the example below.

Bits Per Pixel: Not applicable

Example:

```
INT i; /* Will hold Loop Index and Error Index */
INT iLineNumber;/* Will hold returned Line Number */
BYTE szFileName[30]; /* Will hold ret'd module name, up to 29
chars */
INT cbFileNameSize; /* Will hold size of szFileName array */
AT_ERRCODE  iCode; /* Will hold returned ImageGear Error Code */
AT_ERRCOUNT  nErrcount;/* Will hold count of errors on error
stack    */
BYTE szBuf[60];/* Will hold zero-terminated string returned by
wsprintf() below */

cbFileNameSize = 30;/* Size of module-name array */
nErrcount = IG_error_check(); /* Get number of errors on stack
*/
for ( i = 0;  i < nErrcount;  i++ )
   {
   /* Get Module Name, Line Number, Error Code, and lValue1,
lValue2: */
   IG_error_get ( i, (LPSTR) &szFileName,
      cbFileNameSize, &iLineNumber, (LPAT_ERRCODE)
      &iCode);
   /* Format error message in szBuf: */
   wsprintf ( szBuf, "Error %d in Module %s at Line
%d", iCode, szFileName, iLineNumber );
/* Display error message in a Message Box, with heading "Error"
: */
   MessageBox ( NULL, szBuf, "Error", MB_OK );
   }
IG_error_clear(); /* Done getting errors, clear the error stack
*/
```

IG_error_set

Group: Library Utility

Declaration:

```
void ACCUAPI IG_error_set (const LPSTR szFileName,
   INT    iLineNumber,
   AT_ERRCODE iCode,
```

```
AT_INT lValue1,
AT_INT lValue2
);
```

Arguments:

szFileName	A long pointer to a string that supplies the name of the module from which the error was generated. It is recommended that you use the _FILE_ constant in this field.
iLineNumber	An integer telling ImageGear from which line the error was set. It is recommended that you use the _LINE_ constant in this field.
iCode	An integer value of type AT_ERRCODE. Set this to the code number of the error that you wish to place on the error stack. If you will be placing an error on the stack that you have defined yourself, be sure to set it to a number that is less than the current value of IG_LAST_ERROR_NUMBER.
lValue1 lValue2	These LONG arguments are available so that you may supply any supporting information about the error. Your application might use these values to decide what to do after setting a particular kind of error.

Description:

This function is called to place an ImageGear error onto the error stack.

One use for this function is with the callback functions. It allows you to write the loading and saving of DIBs and individual raster lines (**IG_load_FD_CB()**, **IG_save_FD_CB_ex()**) to the handle. Each of these functions call your callback functions in order to supply data, such as width, height, and Bits Per Pixel, to ImageGear. Your callback functions (which must be of type **LPFNIG_DIB_GET**, **LPFNIG_RASTER_GET**, **LPFNIG_DIB_CREATE**, and **LPFNIG_RASTER_SET**), must return an error count to **IG_load_FD_CB()** and IG_save_FD_CB(). You would do this by calling **IG_error_check()** after each raster is loaded or saved. If you wanted to terminate the load or save, you could use IG_error_set() to place the ImageGear error of your choosing, upon the stack.

TIP: If you are setting an error code that you have defined yourself, you must make sure that it has a value less than ImageGear's IG_LAST_ERROR_NUMBER. As the defined value of IG_LAST_ERROR_NUMBER may change in the future, you should always use the constant name during your check, not the integer value. For example:

```
#define MYERROR1 (IG_LAST_ERROR_NUMBER - 1)
#define MYERROR2  (IG_LAST_ERROR_NUMBER - 2)
```

Bits Per Pixel:

Not applicable

Example:

```
#define MYERR_BAD_RASTER (IG_LAST_ERROR_NUMBER - 1)
 …
AT_ERRCODE  iCode;/* Initialize an integer of type AT_ERRCODE
*/
AT_ERRCOUNT nErrcount; /* Count of returned errors on stack */
```

```
LPFNIG_DIB_GET MyDIBGet;/* Will provide DIB header  */
LPFNIG_RASTER_GET MyRasterGet;  /* Will provide each raster */
INT fd;      /* File Descriptor handle    */
DWORD dwPrivate[10];/* Some private data     */
BYTE szFileName[30];/* Will set name of guilty module    */
nErrcount = IG_save_FD_CB ( fd, 1, 0, IG_SAVE_BMP_UNCOMP,
MyRasterGet, MyDIBGet, (LPVOID) &dwPrivate );
 …
AT_ERRCOUNT  ACCUAPI MyRasterGet (LPVOID lpPrivate, LPAT_PIXEL
lpRast, AT_PIXPOS lYpos, DWORD dwPrivate)
{
 …
   /* if error found in raster line just read */
   If (bytes_read!=bytes_expected){
      nErrcount = IG_error_set(FILE,LINE,
      MYERR_BAD_RASTER,bytes_read, bytes_expected);
   }
 …
return IG_error_check();
}
```

IG_file_IO_register

Group: Callback

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_file_IO_register (LPFNIG_READ
lpfnReadFunc,
   LPFNIG_WRITE lpfnWriteFunc,
   LPFNIG_SEEK lpfnSeekFunc
   );
```

Arguments:

lpfnReadFunc	Far pointer to your function to be called for **READs**
lpfnWriteFunc	Far pointer to your function to be called for **WRITEs**
lpfnSeekFunc	Far pointer to your function to be called for **SEEKs**

Description:

This function registers your own functions to be called to do Reads, Writes, and Seeks during image file transfers. An argument should be `NULL` if you want ImageGear to perform that operation. See also the descriptions for typedefs **`LPFNIG_READ`**, **`LPFNIG_WRITE`**, and **`LPFNIG_SEEK`**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
LPFNIG_READ MyReadFunc;      /* To be called for file READs */
{
HIGEAR  hIGear;     /* Will hold HIGEAR handle of image  */
 …
IG_file_IO_register ( MyReadFunc, NULL, NULL );/* Register it */
 …
IG_load_file ( “picture.bmp”, &hIGear );
 …
}
/* This will be called for each read during the above Load:   */
LONG  ACCUAPI MyReadFunc ( LONG fd, LPBYTE lpBuffer,
   LONG lNumToRead )
{
LONG nNumActuallyRead;
 …   /* May transfer bytes to buffer in any way     */
return  nNumActuallyRead; /* Return count, or -1 for error  */
}
```

IG_fltr_compressionlist_get

Group: Format filters

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_fltr_compressionlist_get (
[IN]   LPAT_DIB lpDIB,
[IN]   AT_MODE nFormatID,
[OUT]  LPAT_MODE lpComprList,
[IN]   UINT   nCListSize,
[OUT]  LPUINT lpnCListCount
       );
```

Arguments:

lpDIB	Pointer to the AT_DIB structure that contains image parameters. If NULL, then this function retursn all possible compressions.
nFormatID	File format identifier of the IG_FORMAT_... constant.
lpComprList	Pointer to the array indicating where to return list of IG_COMPRESSION_...compression constants
nCListSize	The size of array lpCompList
lpnCListCount	Pointer indicating where to receive the actual number of compression constants.

Description:

This function allows you to get information about which compressions are applicable when saving an image with parameters given in the lpDIB header into the file format specified by identifier nFormatID. If lpDIB is NULL, this function returns all supported compressions by this format filter. When lpComprList is NULL, this function returns number of compressions through the lpnCListCount parameter. If lpCompList is not NULL, then this function copies the correspondent compression constants (IG_COMPRESSION_...) into this array. nCListSize specifies the size of lpComprList if it is not NULL.

See also the section Using Format Filters API for Image Saving.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear; /* Will hold handle returned by IG_load_file*/
AT_ERRCOUNT nErrCount;  /* Count of errs on stack upon ret from
func*/
UINT   nCount;
LPAT_DIB lpDIB;
LPAT_MODE lpCompList;

/* Figure out compressions that given HIGEAR is possible to save
into TIFF format */
nErrCount = IG_image_DIB_pntr_get( hIGear, &lpDIB );

if( nErrCount==0 )
{
   /* get number of compressions */
   IG_fltr_compressionlist_get( lpDIB, IG_FORMAT_TIF, NULL, 0,
&nCount );
   lpCompList = (LPAT_MODE)malloc( nCount*sizeof(AT_MODE) );
   if( lpCompList!=NULL )
   {
```

```
   IG_fltr_compressionlist_get( lpDIB, IG_FORMAT_TIF,
lpCompList, nCount, &nActual );
       …
      free( lpCompList );
   }
}
```

IG_fltr_compressionlist_get_ex

Group: Format Filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_compressionlist_get_ex(
const HIGDIBINFO hDIB,
AT_MODE nFormatID,
LPAT_MODE lpComprList,
UINT nCListSize,
LPUINT lpnCListCount
);
```

Arguments:

hDIB	Handle of DIB info object that contains image parameters. If NULL, then this function returns all possible compressions.
nFormatID	File format identifier of the IG_FORMAT_... constant.
lpComprList	Pointer to the array indicating where to return list of IG_COMPRESSION_...compression constants
nCListSize	The size of array lpCompList
lpnCListCount	Pointer indicating where to receive the actual number of compression constants.

Description:

This function allows you to get information about which compressions are applicable when saving an image with parameters given in the `hDIB` info object into the file format specified by identifier `nFormatID`.

If `hDIB` is `NULL`, this function returns all supported compressions by this format filter.

When `lpComprList` is `NULL`, this function returns number of compressions through the `lpnCListCount` parameter. If `lpCompList` is not `NULL`, then this function copies the correspondent compression constants (`IG_COMPRESSION_...`) into this array.

`nCListSize` specifies the size of `lpComprList` if it is not `NULL`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGDIBINFO hDIB;        /* DIB info handle of image */
UINT nCount, nActual;   /* Number of compressions */
LPAT_MODE lpCompList;   /* List of compressions */
/* Get list of compressions that can be used when
   saving the given image into TIFF format */
nErrcount = IG_image_DIB_info_get(hImage, &hDIB);
nErrcount = IG_fltr_compressionlist_get_ex(hDIB,
    IG_FORMAT_TIF, NULL, 0, &nCount);
lpCompList = (LPAT_MODE) malloc(nCount * sizeof(AT_MODE));
nErrcount = IG_fltr_compressionlist_get_ex(hDIB,
    IG_FORMAT_TIF, lpCompList, nCount, &nActual);
/* lpCompList now contains list of compressions */
IG_DIB_info_delete(hDIB);
free(lpCompList);
```

IG_fltr_ctrl_get

Group: Format filters

Declaration:

```
AT_ERRCODE  EXPORT   ACCUAPI IG_fltr_ctrl_get(
   DWORD  dwFormatID,
   const LPCHAR lpcsCtrlName,
   AT_BOOL bGetDefault,
   LPAT_MODE lpnValueType,
   LPDWORD lpdwValueSize,
   LPVOID lpBuffer,
   DWORD  dwBufferSize
    );
```

Arguments:

FormatID	Specifies ImageGear filter you want to get the info about. Set it using `IG_FORMAT_` constants.
lpcsCtrlName	Specifies the name of control parameter you want to get the info about. The list of names of supported control parameters can be get using `IG_fltr_ctrl_list()`.
bGetDefault	Each control parameter has default value and this parameter can be used to select what to return through `lpBuffer` parameter. If `TRUE` then default value of control parameter is returned through `lpBuffer`, but in other case current value is returned.
lpnValueType	Returns the value type of control parameter.
lpdwValueSize	Return the size of memory buffer that should be allocated for `lpBuffer` to completely fit returned value. If `NULL` then this parameter is ignored.
lpBuffer	Buffer where to return control parameter value.
dwBufferSize	The size of memory allocated for `lpBuffer` pointer. Please note that if it is less than value returned through `lpdwValueSize` then `lpBuffer` contains not all bytes of control parameter value.

Description:

This function allows you to get full information about every control parameter supported by ImageGear filter. It returns current value of `lpcsCtrlName` control parameter for `dwFormatID` filter (set it as `IG_FORMAT_` constant) through `lpBuffer`. Also it returns the type of this control parameter value (`lpnValueType`). If you want to get default control parameter value set `bGetDefault` argument to `TRUE`. Note that your application is responsible for allocating free necessary memory for `lpBuffer` through `dwBufferSize` argument.

To change the current value of control parameter use function **`IG_fltr_ctrl_set()`**.

See also the section Using Format Filters API for Filter Control.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
...
/* setup JPEG decimation control parameter */
so.nDecimationIndex = GetPrivateProfileInt( lpszSaveSection,
lpszDecimationIndex,
100, lpszIniFileName);
```

```
   if( (so.nDecimationIndex<0) ||
(so.nDecimationIndex>=(sizeof(_JPG_decimate_array)/
sizeof(WORD))) )
   {
      WORD       decim;
      INT         i;
      so.nDecimationIndex = 0;
IG_fltr_ctrl_get( IG_FORMAT_JPG, "DECIMATION_TYPE", FALSE,
NULL, NULL, &decim, sizeof(decim) );
      for( i = 0; i<sizeof(_JPG_decimate_array)/sizeof(WORD);
i++ )
         if( decim==_JPG_decimate_array[i] )
         {
            so.nDecimationIndex = i;
            break;
         }
   }
...
```

IG_fltr_ctrl_list

Group: Format filters

Declaration:

```
AT_ERRCODE  EXPORT   ACCUAPI  IG_fltr_ctrl_list (
   DWORD dwFormatID,
   LPUINT lpnCount,
   LPVOID lpArray,
   DWORD dwArraySizeInBytes
   );
```

Arguments:

FormatID	Specifies ImageGear filter you want to get the info about. Set it using `IG_FORMAT_` constants (See `accucnst.h` file).
lpnCount	Returns the number of supported control parameters for requested filter.
Array	Returns the array of control parameters for requested filter. Application is responsible to allocate memory for `lpArray` buffer but IG fills each element of this array with pointer to static string. Application is not allowed to change memory referenced by any of those string pointers.
ArraySizeInBytes	Provides the size (in bytes) of memory allocated for `lpArray` pointer.

Description:

This function allows you to get the list of control parameters for each ImageGear filter you specify using `FormatID` argument.

See also the section Using Format Filters API for Filter Control.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_fltr_ctrl_set

Group: Format filters

Declaration:

```
AT_ERRCODE  EXPORT   ACCUAPI  IG_fltr_ctrl_set(
   DWORD  dwFormatID,
   const LPCHAR lpcsCtrlName,
   LPVOID lpValue,
   DWORD  dwValueSize
);
```

Arguments:

FormatID	Specifies ImageGear filter wich control parameter you want to set. Set it using `IG_FORMAT_` constants.
lpcsCtrlName	Specifies the name of control parameter you want to set. The list of names of supported control parameters can be get using **`IG_fltr_ctrl_list()`**.
lpValue	Specifies new value to set for control parameter with name `lpcsCtrlName`. General rule for format is following. If size of value is <= `sizeof(DWORD)` then `lpValue` should contain actual value but in other case it should contain pointer to value.
dwValueSize	Specifies the size (in bytes) of control parameter value.

Description:

This function allows you to set the control parameter value specified by `lpcsCtrlName` for the format filter specified by `FormatID`.

See also the section Using Format Filters API for Filter Control.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
...
   {
   case IDCANCEL:
      EndDialog( hWnd, FALSE );
      break;

   case IDOK:
/* close the dialog box when the OK button is pressed */
   nIndex = SendDlgItemMessage( hWnd, IDC_LIST, CB_GETCURSEL,
0, 0 );
    IG_fltr_ctrl_set( IG_FORMAT_JPG, "LOAD_SCANS",
(LPVOID)(LONG)(nIndex), sizeof(nIndex) );
   EndDialog(hWnd, TRUE);
   break;
   }
   break;
...
```

IG_fltr_detect_FD

Group: Format filters

Declaration:

```
AT_ERRCOUNT IG_fltr_detect_FD(
[IN]   AT_INT  fd,
[IN]   LONG   lOffset,
[OUT]  LPAT_MODE lpFileType
);
```

Arguments:

fd	The file descriptor handle of the file in which to load the data.
lOffset	The offset, in bytes, to the file in which the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. `lOffset` is usually 0.
lpFileType	The pointer indicating where to return file format identifier.

Description:

This function detects the file format, specified by given file descriptor starting from the `lOffset` position in the file. It can be either `IG_FORMAT_UNKNOWN` if the format is unknown, or another `IG_FORMAT_`… value defined in `accucnst.h`.

See also the section Detecting the File Format.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
AT_ERRCOUNT nErrCount = IGE_SUCCESS;/* will hold returned error
count */
AT_MODE nFormatID;
INT    fd;

fd = _lopen ( "image.tiff", OF_READ );
 …
nErrCount = IG_fltr_detect_FD( fd, 0, &nFormatID );
if( nFormatID==IG_FORMAT_TIF )
{
    …
}
_lclose( fd );
```

Sample: **None**

IG_fltr_detect_file

Group: Format filters

Declaration:

```
AT_ERRCOUNT   IG_fltr_detect_file(
[IN]     const LPSTR  lpszFileName,
[OUT]    LPAT_MODE    lpFileType
   );
```

Arguments:

lpszFileName	The file name of image format you wish to detect.
lpFileType	A pointer indicating where to return the file format identifier.

Description:

This function detects the file format specified by a given filename. It can be either `IG_FORMAT_UNKNOWN` if the format is unknown, or another `IG_FORMAT_`… value defined in `accucnst.h`.

See also the section Detecting the File Format.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
AT_ERRCOUNT nErrCount = IGE_SUCCESS;/* will hold returned error
count */
AT_MODE nFormatID;

 …
nErrCount = IG_fltr_detect_file( "image.tiff", &nFormatID );
if( nFormatID==IG_FORMAT_TIF )
{
    …
}
```

Sample: **GUI**

IG_fltr_detect_mem

Group: Format filters

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_fltr_detect_mem(
[IN]   VOID FAR* lpImage,
[IN]   AT_UINT   nSize,
[OUT]  LPAT_MODE lpFileType
       );
```

Arguments:

lpImage	A far pointer to the memory location of an image file that is currently in memory.
nSize	Size of image in memory.
lpFileType	A pointer indicating where to return the file format identifier.

Description:

This function detects the format of an image located in the memory specified by the pointer `lpImage`. It can be either `IG_FORMAT_UNKNOWN` if the format is unknown, or another `IG_FORMAT_…` value defined in `accucnst.h`.

See also the section Detecting the File Format.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
AT_MODE nFormatID;
INT    fd;
LPBYTE lpPtr;
AT_UINT   nSize;
fd = _lopen ( "image.tiff", OF_READ );
 …
/* load file into memory */
 …
_lclose( fd );
nErrCount = IG_fltr_detect_mem( lpPtr, nSize, &nFormatID );
if( nFormatID==IG_FORMAT_TIF )
{
    …
}
```

Sample: **None**

IG_fltr_formatlist_get

Group: Format filters

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_fltr_formatlist_get(
[IN]   DWORD  dwFlags,
[OUT]  LPAT_MODE lpFormatList,
[IN]   UINT   nFListSize,
[OUT]  LPUINT lpnFListCount
       );
```

Arguments:

dwFlags	Any combination of IG_FLTR_... flags described in the **IG_fltr_info_get()** function, which specifies the format filter properties.
lpFormatList	An array in which to return the list of IG_FORMAT_... constants. If this value is NULL, the total number of formats is returned through the lpnFListCount parameter.
nFListSize	The size of lpFormatList array if it is not NULL.
lpnFListCount	If lpFormatList is not NULL, then the number of copied format identifiers are returned. Otherwise, the total number of formats is returned.

Description:

This function searches and returns the list of supported format filters with features specified by the dwFlags parameter. Possible values include any combination of IG_FLTR_... constants defined in the IG_fltr_info_get function. This function searches for filters that support ALL requested features (not just some of them).

The result is a list of IG_FILTER_... values returned through the second parameter, lpFormatList. The third parameter specifies its size if its value not NULL. The lpnFListCount parameter returns number of copied values if lpFormatList is not NULL and the total number of formats if it is NULL.

In general, to obtain a complete list of formats, you can call this function twice; the first time to get the number of values to allocate the lpFormatList array of the necessary size, and the second time to receive the actual values.

See also the section Getting Information and Sorting Images in Chapter Using ImageGear.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_ERRCOUNT nErrCount; /* Count of errs on stack upon ret from
func*/
UINT   nCount;
LPAT_MODE lpFormatList;

/* Get list filters that support detect and saving */
/* get total count */
nErrCount =
IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERT
SUPPORT, NULL, 0, &nCount );

if( nErrCount==0 )
{
```

```
    /* allocate memory */
    lpFormatList = (LPAT_MODE)malloc( nCount*sizeof(AT_MODE) );
    if( lpFormatList!=NULL )
    {

IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERT
SUPPORT, lpFormatList, nCount, NULL );
        …
       free( lpFormatList );
    }
}
```

Sample: **GUI**

IG_fltr_formatlist_sort

Group: Format filters

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_fltr_formatlist_sort(
[IN/OUT]   LPAT_MODE    lpFormatList,
[IN]       UINT         nFListSize
);
```

Arguments:

lpFormatList	An array of `IG_FORMAT_…` constants..
nFListSize	The size of `lpFormatList` array.

Description:

This auxiliary function sorts formats in alphabetic order based on the short format name returned through the third parameter of function **`IG_fltr_info_get()`**.

See also the section Getting Information and Sorting Images.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_ERRCOUNT nErrCount; /* Count of errs on stack upon ret from
func */
UINT   nCount;
LPAT_MODE lpFormatList;
```

```
/* Get list filters that support detect and saving */

/* get total count */
nErrCount =
IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERT
SUPPORT, NULL, 0, &nCount );

if( nErrCount==0 )
{
   /* allocate memory */
   lpFormatList = (LPAT_MODE)malloc( nCount*sizeof(AT_MODE) );
   if( lpFormatList!=NULL )
   {

IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERT
SUPPORT,
    lpFormatList, nCount, NULL );
      /* sort formats in alphabetic order */
      IG_fltr_formatlist_sort( lpFormatList, nCount );
       …
      free( lpFormatList );
   }
}
```

Sample: **GUI**

IG_fltr_ICC_callback_get

Group: Format filters

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_fltr_ICC_callback_get(
   LPVOID        *lplpPrivate,
   LPAFT_IG_ICC_GET_CB      *lplpfnGetCB,
   LPAFT_ANY         *lplpfnReserved1,
   LPAFT_ANY         *lplpfnReserved2
   );
```

Arguments:

lplpPrivate	Private callback data.
lplpfnGetCB	GET callback function.
lplpfnReserved1	Reserved.
lplpfnReserved2	Reserved.

Description:

This function returns the current settings for callbacks that are used for reading and writing ICC profiles. It allows you to retrieve the settings of **LPAFT_IG_ICC_GET_CB** callback function.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_fltr_ICC_callback_set

Group: Format filters

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_fltr_ICC_callback_set(
   LPVOID         lpPrivate,
   LPAFT_IG_ICC_GET_CB      lpfnGetCB,
   LPAFT_ANY          lpfnReserved1,
   LPAFT_ANY          lpfnReserved2
   );
```

Arguments:

lpPrivate	Private callback data.
lpfnGetCB	GET callback function.
lpfnReserved1	Reserved.
lpfnReserved2	Reserved.

Description:

This function allows you to register your ICC callback **LPAFT_IG_ICC_GET_CB** function. It registers callbacks for reading and writing ICC profiles during loading and saving of image files.

- ICC profile reading. Use any image loading function to load an image. As soon as the format filter encounters an ICC profile, it calls the callback function. The ICC profile is provided in the standard ICC format, as a byte array. The toolkit owns the buffer, so the application shall not delete it. If the application needs to use the ICC profile after exiting the callback, it shall copy it to its own buffer.
- ICC profile writing. Use any image saving function to save an image. If the format filter supports ICC profile writing, it calls the callback function before writing the profile. If the HIGEAR being saved has an ICC profile attached to it, format filter ignores this profile and writes the profile obtained from the callback. The ICC profile shall be provided in the standard ICC format, as a byte array. The application owns the buffer, and is responsible for deleting it.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_fltr_info_get

Group: Format filters

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_fltr_info_get(
[IN]   AT_MODE nFormatID,
[OUT]  LPDWORD lpdwInfoFlags,
[OUT]  LPCHAR lpShortName,
[IN]   DWORD  dwSNameSize,
[OUT]  LPCHAR lpFullName,
[IN]   DWORD  dwFNameSize,
[OUT]  LPCHAR lpDefExt,
[IN]   DWORD  dwDefExtSize
       );
```

Arguments:

nFormatID	File format identifier of the IG_FORMAT_... constant.
lpdwInfoFlags	Pointer indicating where to receive the format flags, which can be a combination of the following flags: IG_FLTR_DETECTSUPPORT IG_FLTR_PAGEREADSUPPORT IG_FLTR_MPAGEREADPSUPPORT IG_FLTR_MPAGEWRITEPSUPPORT IG_FLTR_PAGEINSERTSUPPORT IG_FLTR_PAGEDELETESUPPORT IG_FLTR_PAGESWAPSUPPORT IG_FLTR_MPDATASUPPORT If the value is NULL, this flag is ignored.
lpShortName	A pointer indicating where to copy the zero-terminated string with the short file format name. If the value is NULL, this flag is ignored.
dwSNameSize	The size of the lpShortName array in bytes.
lpFullName	A pointer indicating where to copy the zero-terminated string with the full file format name. If the value is NULL, this flag is ignored.
dwFNameSize	The size of array lpFullName in bytes.
lpDefExt	A pointer indicating where to copy the zero-terminated string with file masks separated by a semicolon (;). If the value is NULL, this flag is ignored.
dwDefExtSize	The size of lpDefExt array in bytes.

Description:

This function returns information about the format filter and its supported features. If lpdwInfoFlags in not NULL then the DWORD value is returned with combination of the following flags:

IG_FLTR_DETECTSUPPORT	Detect is supported
IG_FLTR_PAGEREADSUPPORT	Page read is supported
IG_FLTR_MPAGEREADPSUPPORT	In combination with IG_FLTR_PAGEREADSUPPORT, this flag means any page loading from a multi-page form
IG_FLTR_PAGEINSERTSUPPORT	Page write is supported
IG_FLTR_MPAGEWRITEPSUPPORT	In combination with IG_FLTR_PAGEINSERTSUPPORT, this flag means the page is inserted into a multi-page form support

IG_FLTR_PAGEDELETESUPPORT	Page delete support from multi-page formats
IG_FLTR_PAGESWAPSUPPORT	Page swap support for multi-page formats
IG_FLTR_MPDATASUPPORT	Supports faster multi-page access by storing private format data (used only with IG_mpi_... and IG_mpf_... API).

This function also returns short format name - usually 3-4 chars (“TIFF”), full format name (“Tagged Image File Format”), and default file extensions separated by “;” (“*.tif;*.tiff”).

See also the section Getting Information and Sorting Images.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_ERRCOUNT SaveImage(
    LPCHAR lpFileName,
    HIGEAR hIGear,
    AT_MODE nFormat,
    AT_MODE nCompression
)
{
    AT_ERRCOUNT nErrCount;
    DWORD  dwFlags;
    CHAR   FName[20];

    nErrCount = IG_fltr_info_get( nFormat, &dwFlags, FName,
sizeof(FName),
         NULL, 0, NULL, 0 );
    if( nErrCount==0 )
    if( (dwFlags&IG_FLTR_PAGEINSERTSUPPORT)!=0 )
    {
        nErrCount = IG_save_file( hIGear, lpFileName,
        nFormat|(nCompression<<16) );
    }else
        printf( "File format %s does not support image saving",
FName );

    return nErrCount;
}
```

Sample: **GUI**

IG_fltr_load_FD_format

Group: Format Filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_load_FD_format(
   AT_MODE nFormat,
   AT_INT fd,
   LONG lOffset,
   UINT nPageNumber,
   UINT nTileNumber,
   LPHIGEAR lphIGear
   );
```

Arguments:

nFormat	File format id to load. If IG_FORMAT_UNKNOWN then usual detect works
fd	File descriptor
lOffset	Offset to image
nPageNumber	Page number to load
nTileNumber	Tile number to load, set to 1 for non-tiled image
lphIGear	Image Gear handle

Description:

This function is similar to `IG_load_FD()` function, but has an additional `nFormat` parameter that specifies the format of the file from which the image should be loaded. If `nFormat = IG_FORMAT_UNKNOWN`, then the usual loading procedure is performed. Otherwise, ImageGear loads the file with specified format filter.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

IG_fltr_load_file

Group: Format filters

Declaration:

```
AT_ERRCOUNT    ACCUAPI    IG_fltr_load_file(
    const LPSTR lpszFileName,
    UINT   nPageNumber,
    LPHIGEAR lphIGear
    );
```

Arguments:

szFileName	Far pointer to FileName (you may include path with filename) of the multipage file from which an image should be loaded into memory.
nPageNumber	The PageNumber of the page-image to load into memory from the `szFileName` file.
hIGear	Far pointer to the HIGEAR object in which to return the ImageGear handle of the image.

Description:

This function loads an image from `nPageNumber` of `szFileName` file into memory and creates a DIB for that image. The handle that ImageGear assigns for this image is returned in the `hIGear` argument. The file named by `szFileName` may be in any format recognized by ImageGear. `IG_fltr_load_file()` will determine the format by inspecting the file's header section. See Chapter File Format Reference.

NOTE: Note that simply loading the file does not cause it to be displayed. Refer to **`IG_dspl_image_draw()`** and related routines, for how to display an image once it is in memory. See also **`IG_load_file_display()`**.

See also the section Using the Format Filters API for Image Loading and Getting Information..

NOTE: Some file formats, such as TXT (ASCII Text), JPEG, and others, may be loaded with additional control, using **`IG_fltr_ctrl_get()`** and **`IG_fltr_ctrl_set()`**. See the description of these functions also in Using Format Filters API for Filter Control section.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR    hIGear     /* Will hold handle returned by
IG_fltr_load_file */
UINT      nPage
AT_ERRCOUNT nErrcount;   /* Count of errs on stack upon ret from
func*/
/* load the selected image */
nErrCount = IG_fltr_load_file(szFile, nPage, &hIGear );
```

IG_fltr_load_file_format

Group: Format filters

Declaration:

```
AT_ERRCOUNT EXPORT  ACCUAPI IG_fltr_load_file_format(
   AT_MODE nFormat,
   const LPSTR lpszFileName,
   UINT   nPageNumber,
   LPHIGEAR lphIGear
   );
```

Arguments:

nFormat	File format ID to load the file. It is specified by such constants as IG_FORMAT_. For the full list of these constants, see the accucnst.h file.
szFileName	Far pointer to the filename (you may include the path with the filename) of the multipage file from which an image should be loaded into memory.
nPageNumber	The PageNumber of the page-image to load into memory from the szFileName file.
hIGear	Far pointer to the HIGEAR object in which to return the ImageGear handle of the image.

Description:

This function is almost the same as the **IG_fltr_load_file()** function, but has an additional nFormat parameter that specifies the format of the file from which the image should be loaded. If nFormat = IG_FORMAT_UNKNOWN then an usual loading procedure is performed. In other case this procedure isn't performed, but ImageGear loads file with specified format filter.

See also the section Using the Format Filters API for Image Loading and Getting Information.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
...
/* Load the image specified on the command line (if any) */

if (lpCmdLine[0])
{
   strcpy( FileName, lpCmdLine );
   /* determine the number of pages */
  IG_fltr_pagecount_file_format( IG_FORMAT_TXT, FileName,
&g_nPageCount );

   /* load the selected image */
  IG_fltr_load_file_format( IG_FORMAT_TXT, FileName,
g_nCurrentPage, &hIGear);
   }
...
```

IG_fltr_metad_callback_get

Group: Format filters

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_fltr_metad_callback_get(
   LPVOID        *lpPrivate,
   LPAFT_IG_METAD_ITEM_SET_CB *lplpfnSetCB,
   LPAFT_IG_METAD_ITEM_ADD_CB *lplpfnAddCB,
   LPAFT_IG_METAD_ITEM_GET_CB *lplpfnGetCB
   );
```

Arguments:

Private	Returns private data associated with metadata callback functions.
SetCB	Returns pointer to callback function of type **LPAFT_IG_METAD_ITEM_SET_CB** that is used for **Set** metadata operation.
AddCB	Returns pointer to callback function of type **LPAFT_IG_METAD_ITEM_ADD_CB** that is used for **Add** metadata operation.
GetCB	Returns pointer to callback function of type **LPAFT_IG_METAD_ITEM_GET_CB** that is used for **Get** metadata operation.

Description:

This function returns current value of callback information that ImageGear uses to pass/receive application data during save/load filter's operations. `NULL` value is possible for any parameter – means that some information is not necessary.

See also **`IG_fltr_metad_callback_set()`**, **`LPAFT_IG_METAD_ITEM_ADD_CB`**, **`LPAFT_IG_METAD_ITEM_GET_CB`**, **`LPAFT_IG_METAD_ITEM_SET_CB`** functions and the section Processing of non-image data through filter callback functions.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

IG_fltr_metad_callback_set

Group: Format filters

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_fltr_metad_callback_set(
   LPVOID        *lpPrivate,
   LPAFT_IG_METAD_ITEM_SET_CB *lplpfnSetCB,
   LPAFT_IG_METAD_ITEM_ADD_CB *lplpfnAddCB,
   LPAFT_IG_METAD_ITEM_GET_CB *lplpfnGetCB
   );
```

Arguments:

Private	New value for private data to be associated with callback functions.
SetCB	New value of callback function of type **LPAFT_IG_METAD_ITEM_SET_CB** that is to be used for **Set** metadata operation.
AddCB	New value of callback function of type **LPAFT_IG_METAD_ITEM_ADD_CB** that is to be used for **Add** metadata operation.
GetCB	New value of callback function of type **LPAFT_IG_METAD_ITEM_GET_CB** that is to be used for **Get** metadata operation.

Description:

This function set new value of callback information that ImageGear uses to pass/receive application data during save/load filter's operations. NULL value is possible for any parameter – means that some information is not necessary.

See also **IG_fltr_metad_callback_get()**, **LPAFT_IG_METAD_ITEM_ADD_CB**, **LPAFT_IG_METAD_ITEM_GET_CB**, **LPAFT_IG_METAD_ITEM_SET_CB** functions and the section Processing of non-image data through filter callback functions.

NOTE: To avoid an application lock up lpfnAddCB metadata callback should always return FALSE if there are no data to send.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
VOID ACCUAPI MetaDataGet(
   LPVOID lpPrivate,     /* Private callback data. */
   LPCHAR ItemName,      /* Name of data item  */
   DWORD ItemID,        /* ID of data item   */
   AT_MODE ItemType,      /* Type of item */

   LPVOID ItemValue,       /* value of item */
   AT_MODE ValueType,       /* type of value */
   DWORD ValueLength,       /* length of value */
   AT_BOOL ReadOnlyValue  /* inform about is value is
changeable or not */
)
{
}
```

```
BOOL ACCUAPI MetaDataSet(
   LPVOID         lpPrivate,     /* Private callback data. */
   LPCHAR         ItemName,       /* Name of data item */
   DWORD          ItemID,         /* ID of data item    */
   AT_MODE        ItemType,       /* Type of item */

   LPVOID         ItemValue,      /* value of item */
   AT_MODE        ValueType,      /* type of value */
   DWORD          ValueLength,    /* length of value */
   AT_BOOL        ReadOnlyValue, /* inform about is value is
changeable or not*/
   LPVOID         *NewItemValue,
   LPAT_MODE      NewValueType,
   LPDWORD        NewValueLength
)
...
BOOL ACCUAPI MetaDataAdd(
   LPVOIDlpPrivate,        /* Private callback data. */
   LPCHAR*ItemName,         /* Name of data item */
   DWORD*ItemID,          /* ID of data item     */
   AT_MODE*ItemType,         /* Type of item */

   LPVOID*ItemValue,        /* value of item */
   AT_MODE*ValueType,        /* type of value */
   DWORD*ValueLength,      /* length of value */
   AT_BOOL*ReadOnlyValue  /* inform about is value is
changeable or not */
)
...
FillAddMetaDList();

   IG_fltr_metad_callback_set( &PrivMetaD, MetaDataSet,
MetaDataAdd, MetaDataGet );
...
```

IG_fltr_metad_update_file

Group: Format filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_metad_update_file(
const LPSTR lpszFileNameSrc,
const LPSTR lpszFileNameDest,
AT_LMODE lFormatType,
UINT nPageNumber
   );
```

Arguments:

lpszFileNameSrc	Source file name.
lpszFileNameDest	Destination file name.
lFormatType	Source file format.
nPageNumber	Page number for metadata updating.

Description:

This metadata updating function creates a new file with exact copy of source file's pixel data and with new metadata. Pixel data is not decoded but copied directly from source to destination file. At the moment, the following file formats supports this feature:

- `IG_FORMAT_TIF`

NOTE: TIFF-JPEG format is not supported by this function.

The function creates a new file that contains copy of source file data with the new metadata for required page. `lFormatType` parameter value should be the same as the source file format type. In a multipage file, the rest of pages are copied verbatim from source to destination file.

The function obtains new metadata from metadata callback functions (**`LPAFT_IG_METAD_ITEM_SET_CB`**, **`LPAFT_IG_METAD_ITEM_ADD_CB`**).

`IG_fltr_metad_update_file()` function can be used as follows:

1. Load necessary page to get metadata;
2. Change metadata (add / delete / change metadata tags or metadata values);
3. Call `IG_fltr_metad_update_file()` function. `nPageNumber` and `lFormatType` parameter values should correspond to the loaded page and source file format.

Destination file will have source file copy with new metadata for required page.

Bits Per Pixel: ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

Example of metadata update function usage is in the **Filter** sample. Use the following steps to test this feature:

1. Open TIF image using File/Open menu item;
2. Add or modify tag(s) using Image/Metadata/Data Structure dialog
3. Create a file with updated metadata using File/Update metadata menu item. Source file name and page number correspond to the last loaded page in the filter sample.

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
nErrcount = IG_fltr_metad_callback_set(
    dwPrivate, lpfnSetCB, lpfnAddCB, NULL);
nErrcount = IG_fltr_metad_update_file(
    "test.jpg", "updated.jpg", IG_FORMAT_JPG, 1);
```

NOTE: All changes made in the Metadata dialog will be lost, if any function that returns metadata is called before calling `IG_fltr_metad_update_file()` (loading a new file, getting file info or preview).

Sample: Filter

IG_fltr_pagecount_FD_format

Group: Format Filters

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_fltr_pagecount_FD_format(
   AT_MODE nFormat,
   AT_INT fd,
   LONG lOffset,
   LPUINT lpPageCount
   );
```

Arguments:

nFormat	File format id to load. If IG_FORMAT_UNKNOWN then usual detect works
fd	File descriptor
lOffset	Offset to image
lpPageCount	return: Number of pages in an image

Description:

This function is similar to the `IG_page_count_get_FD()` function, but has an additional parameter, `nFormat`, which specifies the file format of the input file. If `nFormat = IG_FORMAT_UNKNOWN`, then the usual autodetection procedure is performed before retrieving the page count. Otherwise, this procedure is not performed, and ImageGear retrieves page count using the specified format filter.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

IG_fltr_pagecount_file_format

Group: Format filters

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_fltr_pagecount_file_format(
   AT_MODE nFormat,
   const LPSTR lpszFileName,
   LPUINT lpPageCount
   );
```

Arguments:

nFormat	The File format ID to load the file. It is specified using constants such as IG_FORMAT_. For a complete list of these constants, see the accucnst.h file.
szFileName	The File Name of image file from which pages should be counted.
PageCount	The number of pages returned by this function.

Description:

This function is similar to the **IG_page_count_get()** function, but has an additional parameter, nFormat, which specifies the file format of the input file. If nFormat = IG_FORMAT_UNKNOWN, then the usual autodetection procedure is performed before retrieving the page count. Otherwise, this procedure is not performed, and ImageGear retrieves page count using the specified format filter.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See example for **IG_fltr_load_file_format()**.

IG_fltr_pagedelete_file

Group: Format filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pagedelete_file(
   const LPSTR lpszFileName,
   AT_MODE nFormatType,
   UINT   nStartPage,
   UINT   nRange
   );
```

Arguments:

szFileName	Far pointer to FileName (you may include path with filename) of the multipage file from which (**nRange**) pages should be deleted.
nFormatType	Specifies the filter ID of the **szFileName** file format. If **IG_FORMAT_UNKNOWN** is passed then the normal detection procedure is performed.
nStartPage	Determines the first page to delete from the **szFileName** file.
nRange	Determines the number of pages to delete, starting from **nStartPage**.

Description:

This function deletes `nRange` pages from the multipage file format, starting with the `nStartPage` page. Not all filters support this procedure. To determine whether the format filter is supported, use the **`IG_fltr_info_get()`** function. If the `IG_fltr_info_get()` function returns the flags `((*lpdwInfoFlags)&IG_FLTR_PAGEDELETESUPPORT)!=`, then the format filter supports the deleting procedure.

See also the section Getting Information and Sorting Images.

NOTE: Currently only TIFF filter supports this procedure.

Bits Per Pixel:

1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_fltr_pageinfo_get

Group: Format filters

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_fltr_pageinfo_get(
   const LPSTR lpszFileName,
   UINT   nPageNumber,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   LPAT_DIB lpDIB
   );
```

Arguments:

lpszFileName	Name of the file for which to get information.
nPageNumber	Page Number of the page in a multi-page file for which to get information.
lpFileType	Far pointer to an AT_MODE variable for which the file type, such as IG_FORMAT_BMP, will be returned.
lpCompression	Far pointer to an AT_MODE variable for which a compression type, such as IG_COMPRESSION_RLE, will be returned
lpDIB	Far pointer to an AT_DIB structure to which other file information, such as width, height, and Bits Per Pixel will be returned.

Description:

This function obtains information about the page specified by the nPageNumber parameter from a named multipage file without actually loading it. The file is opened and the header is inspected. The file must exist.

See also the section Getting Information and Sorting Images.

NOTE: This function is almost the same as the **IG_info_get_ex()** function, but returns information about **any** multipage file page specified by nPageNumber, not just the first page.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_fltr_pageswap_file

Group: Format filters

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_fltr_pageswap_file(
   const LPSTR lpszFileName,
   AT_MODE nFormatType,
   UINT   nPage1,
   UINT   nPage2
   );
```

Arguments:

szFileName	Far pointer to filename (you may include path with filename) of the multipage file in which two pages should be swapped.
nFormatType	Specifies the filter ID of the **szFileName** file format. If **IG_FORMAT_UNKNOWN** is passed, then the normall detection procedure is performed.
nPage1	Number of page 1 to swap with page 2.
nPage2	Number of page 2 to swap with page 1.

Description:

This function swaps two pages (specified by `nPage1` and `nPage2` accordingly) in the `szFileName` multipage file. This operation is supported by the filters for which the **`IG_fltr_info_get()`** function returns the `((*lpdwInfoFlags)&IG_FLTR_PAGESWAPSUPPORT)!=0` flags.

NOTE: Currently only TIFF filter supports this procedure.

See also the section Getting Information and Sorting Images.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_fltr_raster_plane_callback_get

Group: Format Filters

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_fltr_raster_plane_callback_get(
LPFNIG_RASTER_PLANE_SET *lplpfnRasterPlaneSetCB,
AT_VOID **lpReserved
   );
```

Arguments:

lplpfnRasterPlaneSetCB	SET callback function
lpReserved	reserved for Get function

Description:

This function allows you to retrieve the settings of raster plane callback `LPFNIG_RASTER_PLANE_SET` function.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
nErrcount = IG_fltr_raster_plane_callback_get(
    &lpfnRasterPlaneSetCB, NULL);
```

IG_fltr_raster_plane_callback_set

Group: Format Filters

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_fltr_raster_plane_callback_set(
LPFNIG_RASTER_PLANE_SET lpfnRasterPlaneSetCB,
AT_VOID *Reserved
   );
```

Arguments:

lpfnRasterPlaneSetCB	SET callback function
Reserved	reserved for Get function

Description:

This function allows you to register your raster plane callback `LPFNIG_RASTER_PLANE_SET` function.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
nErrcount = IG_fltr_raster_plane_callback_set(
    lpfnRasterPlaneSetCB, NULL);
```

IG_fltr_save_file

Group: Format filters

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_fltr_save_file(
   HIGEAR hIGear,
   const LPSTR lpszFileName,
   AT_LMODE lFormatType,
   UINT   nPageNumber,
   AT_BOOL bOverwrite
   );
```

Arguments:

hIGear	HIGEAR handle of the image to save
lpszFilename	Far pointer to the filename (you may include path with filename) to save.
lFormatType	A constant such as IG_SAVE_TIF_UNCOMP or IG_SAVE_BMP_RLE, indicating the format in which to save the image. It also specifies the compression scheme if applicable. See the accucnst.h file for a list of all IG_SAVE_... constants.
nPageNumber	Specifies the page number of the page inserted into a multi-page file.
bOverwrite	AT_BOOL parameter. If TRUE - the file is overwritten during the save procedure, if FALSE - the file isn't overwritten during inserting page procedure.

Description:

This function stores the image rectangle of the image referenced by **hIGear** to disk using the name lpszFilename. lFormatType is used to set the format and compression (if applicable) of the output file. If you want to have ImageGear use the file extension provided in your filename string (lpszFilename) to determine the file format in which to save the file, set lFormatType = IG_SAVE_UNKNOWN.

NOTE: This function is similar to the **IG_save_file()** function, but it allows you to insert a new page in multi-page file as either the end page or the page with a given nPageNumber number. The second bOverwrite parameter determines whether or not the **lpszFilename** file should be overwritten during the save procedure. (yes=TRUE) (no=FALSE).

See also the section Using Format Filters API for Image Saving.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
/* The code example for using “File|Save As” menu option:  */

case ID_FILE_SAVEAS:
   memset( &SFInfo, 0, sizeof(SFInfo) );

   IG_image_DIB_pntr_get( hIGear, &lpDIB );
/* get a file name from the save common dialog box */
    if( IG_gui_save_page_dlg( hWnd, NULL, lpDIB, NULL, 0,
&SFInfo ) )
    {
    SetCursor(LoadCursor(NULL, IDC_WAIT));

/* save the image in the selected image file format    */
/* of.nFilterIndex - 1 is a zero based offset into the */
/* save filter list, defined in common.c */
err_count = IG_fltr_save_file(hIGear, SFInfo.FileName,
SFInfo.nFormat|(SFInfo.PageInfo.nCompression<<16),
SFInfo.PageInfo.nPage, SFInfo.bOverwrite );

   repaint_image_and_error_check(hWnd);

   SetCursor(LoadCursor(NULL, IDC_ARROW));
    }
break;
```

IG_fltr_savelist_get

Group: Format filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_savelist_get (
```

```
[IN]   LPAT_DIB lpDIB,
[IN]   LPAT_MODE lpnFilterList,
[IN]   UINT   nFListSize,
[OUT]  LPAT_LMODE lpSaveList,
[IN]   UINT   nSListSize,
[OUT]  LPUINT lpnSListCount
)
```

Arguments:

lpDIB	A pointer to the AT_DIB structure that contains the image parameters. If the value is not NULL, this function contains a list of the IG_FORMAT_... constants that defines the formats you can use. If the value is NULL, then all currently supported file formats can be used.
lpnFilterList	This parameter is a pointer to the list of format identifiers (like IG_FORMAT_JPG), which will be used in the save list. If this parameter is NULL, then all currenhtly existing formats will be used.
nFListSize	An array containing the number of elements if lpnFilterList is not NULL.
lpSaveList	An array containing the returned save constants created by combining the format identifier from the lpnFilterList list and the compression format that applies to the image with the parameters stored in lpDIB. If this value is NULL, the total number of records is returned through the lpnSListCount parameter.
nSListSize	The size of the lpSaveList array.
lpnSListCount	If the lpSaveList array is not NULL, this parameter returns the number of copied values. If lpSaveList is NULL, this parameter returns the total number of records.

Description:

This function prepares a list of values (nFormat|(nCompression<<16)), where nFormat is the format identifier IG_FORMAT_... and nCompression is the compression constant IG_COMPRESSION_... defined in the accucnst.h header file.

If the value is not NULL, nFormat values are taken from either lpnFilterList or from the list of all available format filters. In any case, all records are sorted alphabetically by short name, returned from the **IG_fltr_info_get()** function, through the third parameter.

Compression constants are obtained using function **IG_fltr_compressionlist_get()** and its corresponding combination is put into lpSaveList for each format identifier in this list. Each value from this list can be passed directly to the function **IG_save_file()** as the third parameter.

If lpSaveList is NULL, this function counts the size of the required lpSaveList array and returns that count in through the last lpnSListCount parameter.

See also the section Using Format Filters API for Image Saving.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;
AT_ERRCOUNT nErrCount;
UINT   nCount;
LPAT_DIB lpDIB;
LPAT_LMODE lpSaveList;
LPCHAR lpFileName;/* file name where to save image */

/* Prepare the list of save constants that can be used to save
given image */
nErrCount = IG_image_DIB_pntr_get( hIGear, &lpDIB );

if( nErrCount==0 )
{
   /* get number of compressions */
   IG_fltr_savelist_get( lpDIB, NULL, 0, NULL, 0, &nCount );
   lpSaveList = (LPAT_LMODE)malloc( nCount*sizeof(AT_LMODE) );
   if( lpSaveList!=NULL )
   {
   IG_fltr_savelist_get( lpDIB, NULL, 0, lpSaveList, nCount,
NULL );
   // some gui that allows to select one save index
    …
   IG_save_file( hIGear, lpFileName, lpSaveList[
nSelectedFormat ] );
      free( lpSaveListList );
   }
}
```

IG_fltr_savelist_get_ex

Group: Format Filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_savelist_get_ex(
const HIGDIBINFO hDIB,
LPAT_MODE lpnFilterList,
UINT nFListSize,
LPAT_LMODE lpSaveList,
UINT nSListSize,
LPUINT lpnSListCount
);
```

Arguments:

hDIB	Handle of DIB info object that contains image parameters. If NULL, then this function returns all possible file formats.
lpnFilterList	This parameter is a pointer to the list of format identifiers (like IG_FORMAT_JPG), which will be used in the save list. If this parameter is NULL, then all currently existing formats will be used.
nFListSize	An array containing the number of elements if lpnFilterList is not NULL.
lpSaveList	An array containing the returned save constants created by combining the format identifier from the lpnFilterList list and the compression format that applies to the image with the parameters stored in hDIB. If this value is NULL, the total number of records is returned through the lpnSListCount parameter.
nSListSize	The size of the lpSaveList array.
lpnSListCount	If the lpSaveList array is not NULL, this parameter returns the number of copied values. If lpSaveList is NULL, this parameter returns the total number of records.

Description:

This function prepares a list of values (nFormat|(nCompression<<16)), where `nFormat` is the format identifier `IG_FORMAT_...` and `nCompression` is the compression constant `IG_COMPRESSION_...` defined in the **`accucnst.h`** header file.

If the value is not `NULL`, `nFormat` values are taken from either `lpnFilterList` or from the list of all available format filters. In any case, all records are sorted alphabetically by short name, returned from the **`IG_fltr_info_get()`** function, through the third parameter.

Compression constants are obtained using function **`IG_fltr_compressionlist_get()`** and its corresponding combination is put into `lpSaveList` for each format identifier in this list. Each value from this list can be passed directly to the function **`IG_save_file()`** as the third parameter.

If `lpSaveList` is `NULL`, this function counts the size of the required lpSaveList array and returns that count in through the last `lpnSListCount` parameter.

See also the section Using Format Filters API for Image Saving.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Get list of formats that can be used when
   saving the given image */
```

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGDIBINFO hDIB;        /* DIB info handle of image */
UINT nCount, nActual;   /* Number of formats */
LPAT_LMODE lpSaveList;  /* List of formats */
nErrcount = IG_image_DIB_info_get(hImage, &hDIB);
nErrcount = IG_fltr_savelist_get_ex(hDIB, NULL, 0,
    NULL, 0, &nCount);
lpSaveList = (LPAT_MODE) malloc(nCount * sizeof(AT_MODE));
nErrcount = IG_fltr_savelist_get_ex(hDIB, NULL, 0,
    lpSaveList, nCount, &nActual);
/* [some GUI to allow selecting a format] */
nErrcount = IG_save_file(hImage, lpFileName,
    lpSaveList[nSelectedFormat]);
IG_DIB_info_delete(hDIB);
free(lpSaveList);
```

IG_FX_blur

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_blur (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const AT_MODE nBlurMode
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nBlurMode	A constant such as IG_BLUR_3 specifying the kernel size to use to perform the blurring. See file accucnst.h for the IG_BLUR_ constants available.

Description:

This function blurs an image. nBlurMode controls the degree of blurring.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been

set to **TRUE**, indicating that a mask HIGEAR should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image    */
AT_RECT rcImageRect;    /* Image's current image rectangle */
/* Blur only the image rect portion, using a 5 x 5 kernel:    */
IG_FX_blur ( hIGear, &rcImageRect, IG_BLUR_5 );
```

IG_FX_chroma_key

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_chroma_key (HIGEAR hIGearFrGrnd,
   LPAT_RECT lpRect,
   HIGEAR hIGearBkGrnd,
   const DOUBLE dblHueCenter,
   const DOUBLE dblHueRange,
   const UINT nSmooth,
   const UINT nThreshold
   );
```

Arguments:

hIGearFrGrnd	HIGEAR handle of image to modify where the specified hue is found
lpRect	Far pointer to an AT_RECT structure specifying the rectangular portion of the image to be processed. Use NULL for the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
hIGearBkGrnd	HIGEAR handle of an image to insert from on hue match.
dblHueCenter	The angle in degrees (in the standard Color Wheel) of the hue to match. 0.0 - 360.0 (360 == 0).
dblHueRange	The range on which to allow (in degrees) either side. 0.0 - 360.0
nSmooth	An integer from 0 to 25 specifying how much to smooth the transition. 0 gives the sharpest edge.
nThreshold	Intensity below which to ignore the hue, and fail the match (0 - 255)

Description:

This function blends two images, inserting the pixel values from hIGearBkGrnd wherever the pixel in hIGearFrGrnd is in the specified hue range. You can control the smoothness of the transitions using argument nSmooth, and you can prevent the hue of dark pixels from being considered, using nThreshold. To determine the proper hue center and hue range, you may want to use the **IG_DIB_pixel_get()** and **IG_IP_RGB_to_hue()** functions.

NOTE: See also **IG_IP_blend_with_LUT()**, **IG_DIB_pixel_get()** and **IG_IP_RGB_to_hue()** functions.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: See the descriptions of **"IG_IP_NR_ROI_mask_associate" described on page -924** and **"IG_IP_NR_ROI_to_HIGEAR_mask" described on page -927** for more details.

TIP: The "background image" must have the same height, width, and bit depth as the "foreground image."

The hue is not an 8-bit HSI. HSI is the name of the color space.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hiGear,/* HIGEAR handle of image to blend into  */
hIGear Bkgrnd;/* HIGEAR handle of image to blend in  */
AT_PIXEL pixelvalue[3];/* 3 bytes for return of an RGB pixel
value */
DOUBLE Hue;/* Hue angle that will be returned */

/* Retrieve the RGB value of the pixel at (10,20) in the HIGEAR
image */
nErrcount = IG_DIB_pixel_get (hIGear, 10, 20, &pixelvalue[0]);
/* Pass the RGB pixel value to IG_IP_RGB_to_hue to convert to
hue angle  */
nErrcount = IG_IP_RGB_to_hue (&pixelvalue[0], &Hue);
/* Pass newly calculated hue angle to chroma_key to combine
images   */
IG_FX_chroma_key ( hIGear, NULL, hIGearBkgrnd, Hue, 10.0, 0, 20
);
```

IG_FX_diffuse

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_diffuse (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const UINT nStrength
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nStrength	An integer from 1 to 16 specifying the amount of diffusion.

Description:

This function diffuses an image by shuffling the positions of pixels. The greater the value of nStrength, the greater the diffusion.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask HIGEAR should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of image */
/* Diffuse the image slightly: */
IG_FX_diffuse ( hIGear, NULL, 3 );
```

IG_FX_emboss

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_emboss (HIGEAR hIGear,
```

```
const LPAT_RECT lpRect,
const DOUBLE dblStrength,
const AT_MODE nCompassDir
);
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dblStrength	the embossing strength. The valid range is from 1.0 to 5.0
ncompassDir	An AT_MODE Compass derection constants (see accucnst.h file): IG_COMPASS_N - North direction IG_COMPASS_NE - North-East direction IG_COMPASS_E - East direction IG_COMPASS_SE - South-East direction IG_COMPASS_S - South direction IG_COMPASS_SW - South-West direction IG_COMPASS_W - West direction direction IG_COMPASS_NW - North-West direction

Description:

This function produces an embossed or 3-D like chiseled-in-stone look to the image. The result looks similar to the engraved face of a coin. The greater the value of dblStrength, the greater the effect will be (higher ridges and lower depressions). The direction in which the image will appear elevated is selected by nCompassDir.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image  */
AT_RECT rcImageRect;/* Image's current image rectangle  */
/* Emboss only the image rect portion: */
IG_FX_emboss ( hIGear, &rcImageRect, 3.0, IG_COMPASS_NE );
```

IG_FX_motion

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_motion (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const UINT nExtent,
   const AT_MODE nDirection
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for the whole image to be processed. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nExtent	Set to a UINT for the amount of motion you would like applied. If nDirection will be set to "S, W, E, N", the correct range for this variable is 2 - 15. If nDirection will be set to "SE, NE, NW, SW", the correct range for this variable is 3 - 22. This variable determines the extent to which the pixels will be "moved" or "smeared", or literally how many pixel lengths each pixel will "move over."
nDirection	An AT_MODE Compass direction constant. Please see accucnst.h file or the descriptin of **IG_FX_emboss()** function for full list of these constants.

Description:

This function makes the image look as though it was moving when the image was captured. The amount of motion depends on the nAmount parameter. A larger nAmount makes the motion appear faster. Use nDirection to select which direction the image appears to be moving toward. lpRect specifies what portion of the image is to be affected.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;  /* HIGEAR handle of image */
/* Blur whole image to imply fast motion to the south-east:  */
IG_FX_motion ( hIGear, NULL, 6, IG_COMPASS_SE );
```

IG_FX_noise

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_noise (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const WORD nType,
   const DOUBLE dblStrength,
   const INT nHitRate,
   const DOUBLE dblSigma,
   const AT_MODE nColorChannel
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nType	An IG_NOISE constant such as IG_NOISE_LINEAR, IG_NOISE_GAUSSIAN
dblStrength	From 0.0 to 127.0, specifying the degree of noise alteration to introduce into affected pixels
nHitRate	Set = 1 to add noise to all pixels, > 1 to skip pixels, only adding noise to some. Larger values cause fewer pixels to be affected. Valid range: 1 to 500
dblSigma	Used with IG_NOISE_GAUSSIAN, range: 0.1-25.0
nColorChannel	An IG_COLOR_COMP_ constant. See file accucnst.h for the full list of these constants.

Description:

This function is used to create noise in an image. This can make an image look older. nHitRate selects how many pixels may have noise introduced into them. dblStrength determines how strongly a pixel's value is to be altered when it is selected to be altered. nType determines the algorithm used to determine the noise alteration.

nHitRate = 1 indicates that almost every pixel should be affected. A value of 100 would indicate that every 100th pixel should be affected. dblStrength = 1.0 indicates that a value of about +1 to -1 should be added to those pixels selected while dblStrength = 50.0 would increase the noise result by adding values in the range of -50 to +50.

Which pixels are altered and the amount of noise to apply are selected randomly.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image   */
/* Add moderate noise to about 5 percent of the pixels: */
IG_FX_noise ( hIGear, NULL, IG_NOISE_LINEAR, 30.0, 20, 1.0,
IG_COLOR_COMP_RGB );
```

IG_FX_pixelate

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_pixelate (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const AT_DIMENSION nXRes,
   const AT_DIMENSION nYRes,
   const AT_MODE nResampleIn,
   const AT_MODE nResampleOut,
   const WORD radius,
   const LPAT_RGB lprgbBkColor
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nXRes	The distance between the new pixels in the horizontal direction.
nYRes	The distance between the new pixels in the vertical direction.
nResampleIn	An IG_RESAMPLE_IN_ constant. These are listed in accucnst.h
nResampleOut	IG_RESAMPLE_OUT_SQUARE or _CIRCLE, the type of result to produce
nRadius	Radius of new “pixels”, in pixels. Only applicable nResampleOut = IG_RESAMPLE_OUT_CIRCLE
LprgbBkColor	If circle, a far pointer to an AT_RGB struct specifying the background color surrounding the output circles.

Description:

This function redraws an image using what appear to be very large pixels. Use nXRes and nYRes to specify how many “apparent pixels” you want in the result.

nResampleIn tells ImageGear what to do with the block of pixels it reads in. It lets you have each output “apparent pixel” computed on the basis of the average, minimum, maximum, or central pixel in the block. Setting this to IG_RESAMPLE_IN_AV will set the value of all pixels in the block to the average all of the pixel values in the block.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount;
nErrcount = IG_FX_pixelate ( hIGear, NULL, 10, 10,
IG_RESAMPLE_IN_AVE,IG_RESAMPLE_OUT_SQUARE, 0, NULL );
```

IG_FX_posterize

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_posterize (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const WORD nLevels,
   const AT_MODE nColorChannel
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying a rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nLevels	Number of unique colors or levels wanted in the resulting image. Valid range: 1-255
nColorChannel	IG_COLOR_COMP_RGB, or use IG_COLOR_COMP_R, _B, or _G to affect only one color channel.

Description:

Posterize reduces the number of actual colors in the image by creating a “stair case” in the palette. The number of steps wanted is specified by nLevels, in the range 1 to 255. nLevels = 255 would cause no effect while nLevels = 20 would cause 20 equally spaced steps to be created.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image  */
/* Use only 50 colors, regardless of how many unique pixel
values:  */
IG_FX_posterize ( hIGear, NULL, 50, IG_COLOR_COMP_RGB );
```

IG_FX_spotlight

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_spotlight (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const AT_PIXPOS nCenterX,
   const AT_PIXPOS nCenterY,
   const AT_DIMENSION nRadius,
   const UINT nDarkenBy,
   const AT_PIXEL nSmoothing
   );
```

Arguments:

hIGear	HIGEAR handle of an image to place the spotlight effect into
lpRect	Far pointer to an AT_RECT struct specifying a rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nCenterX	X coordinate of the center of the circle to receive a spotlight effect; 0 to Width - 1
nCenterY	Y coordinate of the center of the circle; 0 to Height - 1
nRadius	Radius of the circle, in pixels; 2 to Height - 2
nDarkenBy	How much to darken the remainder of the image
nSmoothing	0 to 255, specifying how much smoothing to apply at the edges; 0 = sharp edges, > 0 = smoother edges.

Description:

This function produces a “spotlight” effect within the circle specified by `nRadius` and (`nCenterX`, `nCenterY`). It leaves the pixels within the circle unchanged while darkening the surrounding pixels by reducing their intensity. Use `nDarkenBy` to specify the reduction in intensity of the surrounding pixels.

Use `nSmoothing` to specify the amount of smoothing at the perimeter of the circle. `nSmoothing = 0` will leave the sharpest edge.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **`AT_RECT`** structure as an argument, so that you can process a rectangular subregion of an image. (*See above.*) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **`TRUE`**, indicating that a mask **`HIGEAR`** should be used with the image. If the flag is set to `TRUE`, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **`AT_RECT`** structure and use the non-rectangular ROI defined by the mask **`HIGEAR`**. To create a non-rectangular region of interest, call **`IG_IP_NR_ROI_to_HIGEAR_mask()`**.

NOTE: Please see the descriptions of **`IG_IP_NR_ROI_mask_associate()`** and **`IG_IP_NR_ROI_to_HIGEAR_mask()`** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image   */
```

```
AT_PIXPOSnXc, nYc;  /* Coords of center of spotlight area */
AT_DIMENSIONnWid, nHi; /* Will receive width and height of image
*/
UINT   nBpp;     /* Bits per pixel, not used */
AT_DIMENSIONnRadius;  /* Radius of spotlight  */
IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp ); /* Get
Wid,Hi */

nXc = nWid / 2;   nYc = nHi / 2;   /* Coords of center */
/* Diameter will be half of smallest dimension:   */
nRadius =  (nWid < nHi) ?  nWid / 4  :  nHi / 4;
/* Darken outside the circle by 40, with some smoothing of
transition:  */
IG_FX_spotlight ( hIGear, NULL, nXc, nYc, nRadius, 40, 10 );
```

IG_FX_stitch

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_stitch (HIGEAR hIGear,
   const LPAT_RECTlpRect,
   AT_MODE   nCompassDir,
   const DOUBLEdblStrength,
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying a rectangular portion of the image to be processed. NULL for the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nCompassDir	An AT_MODE Compass direction constant. Please see accucnst.h file or the descriptin of **IG_FX_emboss()** function for full list of these constants.
dblStrength	0.0 to 5.0 (A very low value will produce a gray screen with no detail.)

Description:

This function produces an effect similar to **IG_FX_emboss()**, except that the output more closely resembles a quilted stitch pattern.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. (*See above.*) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the

image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;         /* HIGEAR handle of image */
AT_RECTrcImageRect;  /* Image's current image rectangle */

/* Emboss only the image rect portion: */
IG_FX_stitch ( hIGear, &rcImageRect, IG_COMPASS_NE, 3.0 );
```

IG_FX_texture

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_texture (HIGEAR hIGear,
   const LPAT_RECTlpRect,
   const HIGEARhTextureImage
   );
```

Arguments:

hIGear	HIGEAR handle of image to apply texture to.
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non rectangular ROI defined by the mask.
hTextureImage	HIGEAR handle of the 8 bit image to be applied to image hIGear to produce textured effect

Description:

This function applies a texture to an image. The texturing image is a small 8-bit grayscale image that is treated as a sign centered image. Pixels in the texture image that are 127 have no effect on the original image, and the farther a texture image pixel is from 127, the greater its effect. The texture image is tiled over the entire original image starting in the top left corner. Any left over is clipped.

NOTE: Sign centered images can be created using the emboss function.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: See the descriptions of **"IG_IP_NR_ROI_mask_associate" described on page -924** and **"IG_IP_NR_ROI_to_HIGEAR_mask" described on page -927** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* Handle of image to be textured */
```

```
HIGEAR  hTextureImage; /* Handle of 8 x 8 pixel 8-bit gray level
texturing image */
/* Apply texture to the whole image: */
IG_FX_texture ( hIGear, NULL, hTextureImage );
```

IG_FX_twist

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_twist (HIGEAR hIGear,
   const LPAT_RECTlpRect,
   const AT_MODEnTwistType,
   const UINTnSquareSize
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT structure specifying the rectangular portion of the image to be processed. Use NULL for whole image. Before ImageGear performs this operation, it determines if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear overrides the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nTwistType	One of the IG_TWIST_ constants: IG_TWIST_90 IG_TWIST_180 IG_TWIST_270 IG_TWIST_RANDOM
nSquareSize	Size in pixels of the regions to which to apply twisting (valid range: 2 to 50).

Description:

This function applies a special effect that makes the image look as if it is being viewed through a shower curtain. The image can still be seen but it is chopped up so that detail is lost.

Each square of pixels of size **nSquareSize** in the image is rotated according to **nTwistType**. If **IG_TWIST_RANDOM** is chosen then each block is rotated one of the directions selected randomly.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned,

ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: See **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except 1bpp indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* Handle of image to apply twisting to  */
/* Apply random twisting to 16 x 16 pixel squares of image:  */
IG_FX_twist ( hIGear, NULL, IG_TWIST_RANDOM, 16 );
```

IG_FX_watermark

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_watermark (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const HIGEAR hWatermark
   );
```

Arguments:

hIGear	HIGEAR handle of image to watermark
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
hWatermark	8-bit sign centered image to use for watermarking

Description:

This function is used to produce a watermark like effect. 8-bit sign centered image **hWatermark** is scaled to match image **hIGear** and is added to it.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* Handle of image to apply watermark to  */
HIGEAR hWMarkImage; /* Handle of watermark image  */
IG_FX_watermark ( hIGear, NULL, hWMarkImage );
```

IG_gctrl_item_by_index_get

Group: Global Parameters

Declaration:

```
AT_BOOL  ACCUAPI  IG_gctrl_item_by_index_get(
    UINT   nIndex,
    LPCHAR CtrlID,
    DWORD  dwIDSize,
    LPAT_MODElpnValType,
    LPVOID lpValue,
    DWORD  dwValSize,
    LPDWORDlpdwValSize,
    LPCHAR lpTextInfo,
    DWORD  dwTextBufSize,
    LPDWORDlpdwTextInfoSize
    );
```

Arguments:

nIndex	IN: Index of global control parameter in the array.
CtrlID	OUT: The name of global control parameter.
dwIDSize	IN: Size of `CtrlID` in bytes.
lpnValType	OUT: The type of global control parameter value.
lpValue	OUT: Buffer where to copy control parameter value.
dwValSize	IN: Size of `lpValue` buffer in bytes.
lpdwValSize	OUT: Actual size of parameter value in bytes.
lpTextInfo	OUT: Buffer where to copy text description of global control parameter.
dwTextBufSize	IN: Size of `lpTextInfo` buffer in bytes.
lpdwTextInfoSize	OUT: Actual size of the text description of parameter.

Description:

This function is like **`IG_gctrl_item_get()`** but returns information about parameter determined by given index in array.

See also the section Working with Global Control Parameters.

Bits Per Pixel: N/A

Return Value:

`TRUE` if global parameter with given name is found, `FALSE` if it is not exist.

IG_gctrl_item_count_get

Group: Global Parameters

Declaration:

```
UINT  ACCUAPI  IG_gctrl_item_count_get();
```

Arguments: None

Description:

This function returns total amount of global parameters in the list.

See also the section Working with Global Control Parameters.

Bits Per Pixel: N/A

Return Value:

Integer value - amount of global parameters in the list.

IG_gctrl_item_get

Group: Global Parameters

Declaration:

```
AT_BOOL  ACCUAPI  IG_gctrl_item_get(
LPCHAR CtrlID,
LPAT_MODE lpnValType,
LPVOID lpValue,
DWORD  dwValSize,
LPDWORD lpdwValSize,
LPCHAR lpTextInfo,
DWORD  dwTextBufSize,
LPDWORD lpdwTextInfoSize
);
```

Arguments:

CtrlID	IN: The name of global control parameter.
lpnValType	OUT: The type of global control parameter value.
lpValue	OUT: Buffer where to copy control parameter value.
dwValSize	IN: Size of `lpValue` buffer in bytes.
lpdwValSize	OUT: Actual size of parameter value in bytes.
lpTextInfo	OUT: Buffer where to copy text description of global control parameter.
dwTextBufSize	IN: Size of `lpTextInfo` buffer in bytes.
lpdwTextInfoSize	OUT: Actual size of the text description of parameter.

Description:

This function return all information about global control parameter identified by name `CtrlID`. Return `TRUE` if parameter with given name is found and `FALSE` if parameter with given name is not exist.

See also the section Working with Global Control Parameters.

Bits Per Pixel: N/A

Return Value:

TRUE if global parameter with given name is found, FALSE if it is not exist.

IG_gctrl_item_id_get

Group: Global Parameters

Declaration:

```
AT_BOOL  ACCUAPI  IG_gctrl_item_id_get(
    UINTnIndex
    LPCHARlpCtrlID,
    UINTnBufSize
    );
```

Arguments:

nIndex	OUT: An index of `lpCtrlID` global control parameter in the parameters array.
lpCtrlID	IN: The name of global control parameter.
nBufSize	IN: Size of `lpCtrlID` buffer in bytes

Description:

This function returns index of the global control parameter with given name in array.

See also the section "Working with Global Control Parameters".

Bits Per Pixel: **N/A**

Return Value:

TRUE if global parameter with given name is found, FALSE if it is not exist.

IG_gctrl_item_set

Group: Global Parameters

Declaration:

```
AT_ERRCODE  ACCUAPI IG_gctrl_item_set(
LPCHAR CtrlID,
AT_MODEnValueType,
LPVOID lpValue,
DWORD  dwValueSize,
LPCHAR lpTextInfo
```

```
);
```

Arguments:

CtrlID	IN: The name of global control parameter in form “<GRPNAME>.<Param name>”.
nValueType	IN: The type of global control parameter value. Constant of kind AM_TID_...
lpValue	IN: Pointer to global control parameter value data.
dwValueSize	IN: Size of global control parameter value data (size of buffer lpValue)
lpTextInfo	IN: Text description of global control parameter value.

Description:

This function sets the value to global control parameter. If parameter with given name ControlID is not exist then it is added. If control parameter with given value is exist and callback function for it is not NULL then callback is called exactly after value is changed by this function. If lpTextInfo is not NULL then previous value of this field is changed to new value but if NULL then it is not changed.

See also the section Working with Global Control Parameters.

Bits Per Pixel: N/A

Return Value:

Return value is a code of last error or NULL if success.

IG_image_bits_per_channel_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_bits_per_channel_get(
HIGEAR hIGear,
AT_INT *lpBitsPerChannel
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpBitsPerChannel	Returned number of allocated bits for each pixel channel

Description:

This function gets the number of bits allocated for each pixel channel in an image: 8, 16 or 32.

NOTE: This is not the same as the bit depth of a channel. The number of bits used for a pixel channel may be less than the number of bits allocated.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
AT_INT nBits;           /* Number of bits per channel */
nErrcount = IG_image_bits_per_channel_get(hImage, &nBits);
```

IG_image_blend_with_alpha

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_blend_with_alpha(
HIGEAR              hSource,
HIGEAR              hAlpha,
LPAT_RECT           blendingArea
);
```

Arguments:

hSource	HIGEAR handle of source image
hAlpha	HIGEAR handle of image to blend into source image
blendingArea	Rectangle of area to perform blending on

Description:

This function blends the color channels of two images according to the alpha channel of the image identified by `hAlpha`. This image must contain an alpha channel. Also, the two images must have the same color space. However, the number of alpha and extra channels may differ.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* HIGEAR handle of source image */
HIGEAR hAlpha;          /* HIGEAR handle of alpha image */
AT_RECT r;              /* Rectangle of area to blend */
r.left = r.top = 0;
r.right = r.bottom = 200;
nErrcount = IG_image_blend_with_alpha(hImage, hAlpha, &r);
```

IG_image_channel_add

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_add(
HIGEAR              hIGear,
AT_UINT             position,
LPCAT_CHANNEL_REF   channel
);
```

Arguments:

hIGear	HIGEAR handle of image to add channel to
position	Index of where channel should be added
channel	Location of channel to add

Description:

This function adds a new channel to an image at the specified position, populating the channel's image data from another channel. `position` is an index into the number of channels starting at 0. `channel` specifies the location of the channel to add, which consists of the HIGEAR handle of the image containing the channel and the index of the channel within that image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Add an "extra" channel to an image */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of destination image */
HIGEAR hExtra;          /* Handle of image to use as extra */
AT_CHANNEL_REF channel; /* Channel to add */
AT_INT nChannels;       /* Number of channels in dest. image */
channel.hImage = hExtra;
channel.uNumber = 0;
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_channel_add(hImage, nChannels, &channel);
nErrcount = IG_image_colorspace_get(hImage, &cs);
```

IG_image_channel_copy_create

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_copy_create(
LPCAT_CHANNEL_REF   source,
LPAT_CHANNEL_REF    copy
);
```

Arguments:

source	Location of source channel to copy
copy	Location of new image with copied channel

Description:

This function creates a new image by copying a single channel from an existing image. Specify in `source` the image and channel index to copy. If the copy is successful, `copy` will contain the HIGEAR handle of a newly allocated image containing the copied channel. The channel index in `copy` will always be set to 0. You are responsible for freeing the new image with IG_image_delete.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Extract the first alpha channel if one exists */
/* Otherwise, extract the first color channel */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF srcChan; /* Channel to copy */
AT_CHANNEL_REF dstChan; /* New image with copied channel */
enumIGColorSpaceIDs cs; /* Source image's color space */
AT_INT nColorChannels;  /* Number of color channels */
nErrcount = IG_load_file("alpha.tif", &hImage);
nErrcount = IG_image_colorspace_get(hImage, &cs);
nColorChannels = IG_util_colorspace_color_count_get(cs);
srcChan.hImage = hImage;
if (IG_util_colorspace_contains_alpha(cs))
    srcChan.uNumber = nColorChannels; /* First alpha channel */
else
    srcChan.uNumber = 0; /* First color channel */
nErrcount = IG_image_channel_copy_create(&srcChan, &dstChan);
hCopy = dstChan.hImage;
nErrcount = IG_save_file(dstChan.hImage, "alpha.bmp",
    IG_SAVE_BMP_UNCOMP);
nErrcount = IG_image_delete(hImage);
```

IG_image_channel_count_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_count_get(
HIGEAR hIGear,
AT_INT *lpChannelCount
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpChannelCount	Channel count of the image

Description:

This function counts the image channels.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

IG_image_channel_depth_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_depth_get(
HIGEAR hIGear,
AT_INT Index,
AT_INT *lpChannelDepth
   );
```

Arguments:

hIGear	HIGEAR handle of image
Index	Channel index to get info on
lpChannelDepth	Bit depth of the chanel

Description:

This function returns the channel bit depth.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

IG_image_channel_depths_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_depths_get(
HIGEAR          hIGear,
AT_INT          ChannelCount,
AT_INT          *lpChannelDepths
   );
```

Arguments:

hIGear	HIGEAR handle of image
ChannelCount	Length of the channel depths array to be filled
lpChannelDepths	Array of the channel depths to be filled

Description:

This function returns an array of the channel bit depths.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

IG_image_channel_depths_change

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_depths_change(
HIGEAR              hIGear,
const AT_INT*       newDepths,
AT_MODE             scaleMode
);
```

Arguments:

hIGear	HIGEAR handle of image
newDepths	New channel depths to set
scaleMode	Mode to use for scaling channel depths - must be one of the following: IG_DEPTH_CHANGE_SCALE IG_DEPTH_CHANGE_NO_SCALE

Description:

This function changes the bit depths of the image channels to the specified depths. If scaling is used, pixel data is scaled to match the new bit depths. Otherwise, pixel data will remain unchanged and will be interpreted as conforming to the new depths.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Alter a 24-bit RGB image to have full 8-bit precision
   for green, but only 1-bit precision for red and blue */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depths[] = { 1, 8, 1 }; /* New channel depths */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_depths_change(hImage, depths,
    IG_DEPTH_CHANGE_SCALE);
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
nErrcount = IG_image_delete(hImage);
```

IG_image_channel_remove

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_remove(
HIGEAR              hIGear,
AT_UINT             position
);
```

Arguments:

hIGear	HIGEAR handle of image
position	Index of channel to remove

Description:

This function removes the specified channel from the source image and shifts the remaining channels without transforming pixel data.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Alter a 24-bit RGB image to use the red channel data
   for both red and blue channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF channel; /* Channel to add */
nErrcount = IG_load_file("test.jpg", &hImage);
channel.hImage = hImage;
channel.uNumber = 0;
nErrcount = IG_image_channel_add(hImage, 3, &channel);
nErrcount = IG_image_channel_remove(hImage, 2);
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

IG_image_channel_update

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_update(
LPCAT_CHANNEL_REF   channelToUpdateWith,
LPCAT_CHANNEL_REF   channelToBeUpdated
);
```

Arguments:

channelToUpdateWith	Channel to use as source channel
channelToBeUpdated	Channel to be replaced by source channel

Description:

This function copies pixel data from one channel to another channel. Each channel is specified by a HIGEAR image handle and the channel's index within that image. The source and destination channels can reside in different images or the same image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Alter a 24-bit RGB image to use the red channel data
for both red and blue channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF chanSrc; /* Channel to update from */
AT_CHANNEL_REF chanDst; /* Channel to update */
nErrcount = IG_load_file("test.jpg", &hImage);
chanSrc.hImage = hImage;
chanSrc.uNumber = 0;
chanDst.hImage = hImage;
chanDst.uNumber = 2;
nErrcount = IG_image_channel_update(&chanSrc, &chanDst);
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

IG_image_channels_combine

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channels_combine(
LPCAT_CHANNEL_REF   channels,
AT_UINT             channelsQty,
enumIGColorSpaceIDs colorSpace,
LPHIGEAR            hIGear
);
```

Arguments:

channels	Array of channel descriptors for channels to combine
channelsQty	Number of channels to combine
colorSpace	Color space of new image
hIGear	Pointer to HIGEAR handle where new image is returned

Description:

This function creates a new image with the color space specified by `colorSpace`. The channels in the new image are copied from the channels specified by `channels`.

NOTE: This function does not do any color space or pixel conversions, it merely merges the pixel data from different channels together.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Make a copy of a 24-bit RGB image in which the
   red and blue channels are swapped */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
HIGEAR hImageCombined;  /* Handle of combined image */
AT_CHANNEL_REF chan[3]; /* Channels to combine */
nErrcount = IG_load_file("test.jpg", &hImage);
chan[0].hImage = hImage;
chan[0].uNumber = 2;
```

```
chan[1].hImage = hImage;
chan[1].uNumber = 1;
chan[2].hImage = hImage;
chan[2].uNumber = 0;
nErrcount = IG_image_channels_combine(chan, 3,
    IG_COLOR_SPACE_ID_RGB, &hImageCombined);
nErrcount = IG_save_file(hImageCombined, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
IG_image_delete(hImageCombined);
```

IG_image_channels_separate

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channels_separate(
HIGEAR              hIGear,
LPAT_CHANNEL_REF    channels,
AT_UINT             channelsQty
);
```

Arguments:

hIGear	HIGEAR handle of image to store separated channels in
channels	Array of channel descriptors for channels to separate
channelsQty	Number of channels to separate

Description:

This function separates channels in an image by creating a new image for each channel. Each new image contains one channel with creates a new image for each channeln array of pages, where each page contains one channel with color space set to grayscale. This method does not do any color space or pixel conversions.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Load an image and save all of its channels as
   separate pages in a multi-page TIFF file */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
```

```
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF *chan;   /* Channels to combine */
AT_INT nChannels;       /* Number of channels in image */
AT_INT i;
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
chan = (AT_CHANNEL_REF *) malloc(nChannels *
sizeof(AT_CHANNEL_REF));
for (i = 0; i < nChannels; i++)
{
    chan[i].hImage = hImage;
    chan[i].uNumber = i;
}
nErrcount = IG_image_channels_separate(hImage, chan,
nChannels);
for (i = 0; i < nChannels; i++)
{
    nErrcount = IG_fltr_save_file(chan[i].hImage, "test.tif",
        IG_SAVE_TIF_UNCOMP, i, !i);
    IG_image_delete(chan[i].hImage);
}
IG_image_delete(hImage);
free(chan);
```

IG_image_colorspace_convert

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_colorspace_convert(
HIGEAR hIGear,
enumIGColorSpaceIDs newColorSpace,
LPCAT_COLORSPACE_CONVERSION_OPTIONS options
);
```

Arguments:

hIGear	HIGEAR handle of image
newColorSpace	Color space to convert to
options	Conversion options (or NULL)

Description:

This function converts an image to the specified color space. Specify the new color space using a value from `enumIGColorSpaceIDs`. Argument `options` controls the conversion flow. You may pass `NULL` for `options` if you don't want to specify any options.

For the detailed information about `AT_COLORSPACE_CONVERSION_OPTIONS` structure and the whole color conversion process please refer to the section Working with Color Space Conversion Engine.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* HIGEAR handle of image */
IG_image_colorspace_convert(hImage, IG_COLOR_SPACE_ID_RGB,
NULL);
```

IG_image_colorspace_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_colorspace_get(
HIGEAR hIGear,
enumIGColorSpaceIDs  *lpColorspace
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpColorspace	Returned color space of the image

Description:

This function gets an image's color space ID.

NOTE: An ImageGear color space ID is actually a combination of values. It contains information about 1) color channels, 2) an alpha channel, and 3) extra channels. Examine the definition of `enumIGColorSpaceIDs` carefully (in `accucnst.h`) and use bitmasks such as `IG_COLOR_SPACE_ID_ColorMask` to isolate the information you want.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
enumIGColorSpaceIDs cs; /* Color space ID */
AT_BOOL bIndexed;       /* Is the image indexed? */
nErrcount = IG_image_colorspace_get(hImage, &cs);
if ((cs & IG_COLOR_SPACE_ID_ColorMask) == IG_COLOR_SPACE_ID_I)
    bIndexed = TRUE;
else
    bIndexed = FALSE;
```

IG_image_compression_type_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_compression_type_get (
   HIGEAR hIGear,
   LPDWORD lpCompression
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpCompression	Far pointer to a DWORD in which will be returned the compression type, such as IG_BI_RLE. The compression types are listed in file ACCUSTD.H.

Description:

This function returns the compression type of the image indicated by **hIGear**. The value will be returned right justified in the 32-bit **DWORD** pointed to by argument **lpCompression**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;         /* HIGEAR handle of image        */
DWORD   dwCompression;  /* Will receive compression type */
IG_image_compression_type_get ( hIGear, &dwCompression );
```

IG_image_convert

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_convert(
   char *lpszSrcFileName,
   char *lpszDstFileName,
   AT_LMODE lFormatType,
   AT_LMODE lCommand,
   AT_LMODE lOptions
   );
```

Arguments:

lpszSrcFileName	The filename of the source file
lpszDstFileName	The filename of the destination file
lFormatType	The format type of output image to save
lCommand	The type of operation to perform (IG_CONVERT_ type constant)
lOptions	Conversion options (bit mask)

Description:

This function allows you to transform image file without decoding it completely and avoiding the need to load it into a memory. The following types of transformations are supported as of this writing:

- Lossless conversion of JPEG compressed image files
- Conversion between PDF and PostScript formats

This method does not process image's metadata. This is responsibility of the user. See filter sample for an example of working with metadata.

Lossless conversion of JPEG compressed image files

This function allows you to apply certain operations on JPEG compressed image files, such as rotation, flipping, etc, without degradation of image quality. It can be used also for lossless conversion between JFIF JPEG and EXIF JPEG file formats, and for adding a thumnbail to a JFIF or EXIF file.

Transformation is done on the DCT coefficients rather than on decompressed pixels, so the lossy decompression/compression stages are not involved.

In contrast, the usual way (to load, rotate and save image) results in significant image degradation, especially when a high compression rate is used.

This function can be useful for converting photographic images between portrait and landscape layouts.

The following lossless operations (`lCommand` argument) are supported:

`IG_CONVERT_NONE`	No conversion
`IG_CONVERT_ROTATE_90`	Rotate 90
`IG_CONVERT_ROTATE_180`	Rotate 180
`IG_CONVERT_ROTATE_270`	Rotate 270
`IG_CONVERT_FLIP_HORIZONTAL`	Flip horizontal
`IG_CONVERT_FLIP_VERTICAL`	Flip vertical
`IG_CONVERT_TRANSPOSE`	Flip about upper left - lower right diagonal
`IG_CONVERT_TRANSVERSE`	Flip about upper right - lower left diagonal

`IG_CONVERT_NONE` mode can be used for converting between JFIF JPEG and EXIF JPEG format, or for adding a thumbnail to the image.

`lOption` parameter is a bit mask. Only one bit flag is supported:

```
IG_CONVERT_OPTION_TRIM = 1
```

An inherent limitation of such conversions is that the source image dimensions must be multiples of the DCT matrix size (typically 8 or 16) to preserve the entire image. If they are not, the remaining pixels at the right and/or bottom are undefined after transform. The function fills them with a mirror projection of the preceeding pixels. This may work well enough for many photographic pictures. If you prefer not to keep the mirrored edge, set `lOption` parameter to `IG_CONVERT_OPTION_TRIM`. With this option set, the function will trim result image dimensions to a multiple of DCT size. In particular:

`IG_CONVERT_NONE`	Resulting image dimensions will not be modified
`IG_CONVERT_ROTATE_90`	Resulting image width can be trimmed
`IG_CONVERT_ROTATE_180`	Resulting image width and height can be trimmed
`IG_CONVERT_ROTATE_270`	Resulting image height can be trimmed
`IG_CONVERT_FLIP_HORIZONTAL`	Resulting image width can be trimmed
`IG_CONVERT_FLIP_VERTICAL`	Resulting image height can be trimmed
`IG_CONVERT_TRANSPOSE`	Resulting image dimensions will not be modified
`IG_CONVERT_TRANSVERSE`	Resulting image width and height can be trimmed

The following formats are supported as both source and destination: JFIF-JPEG, EXIF-JPEG. Only Lossy and Progressive compressions are supported. If any other format is used for either source or destination, the function will return an error.

If `lFormatType == IG_FORMAT_UNKNOWN`, the function will recognize source file format and use it for the destination file to save.

This function also converts the image's thumnbail, if it is present. If the thumnbail is JPEG compressed, it will be converted without degradation of quality. If the source image does not contain a thumnbail, and destination filter's "`SAVE_THUMBNAIL`" control parameter is set to `TRUE`, the function will create a thumbnail from the source image.

Conversion between PDF and PostScript formats

This method can also be used for conversion of the entire document from **PDF** to **PostScript** or back. The method's parameters should be set as follows to achieve that:

lpszSrcFileName	Name of the input PDF or PS document to convert
lpszDstFileName	Name of the output PDF or PS document
lFormatType	Save format, either `IG_FORMAT_PDF`, or `IG_FORMAT_POSTSCRIPT`
lCommand	`IG_CONVERT_NONE`
lOptions	Not used, set to `0`

Bits Per Pixel: All

Return Value:

Number of errors occurred during the function call.

IG_image_create

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create(
HIGDIBINFO hDIB,
HIGEAR *lphIGear
   );
```

Arguments:

hDIB	DIB info object with parameters used to create image
lphIGear	Returned HIGEAR handle of created image

Description:

This function creates a new image according to DIB information stored in a DIB info object. Pixel data is allocated and initialized to black.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Create a new image with the same parameters as
   an existing image */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGDIBINFO hDIB;        /* DIB info handle of image */
HIGEAR hImageCreated;   /* Handle of created image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIB);
nErrcount = IG_image_create(hDIB, &hImageCreated);
/* ... */
IG_DIB_info_delete(hDIB);
IG_image_delete(hImageCreated);
```

IG_image_create_alpha

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_alpha (HIGEAR hIGear,
HIGEAR hIBackgrnd, AT_MODE nCreateMode );
```

Arguments:

hIGear	HIGEAR handle of image
hIBackgrnd	HIGEAR handle to a background image
nCreateMode	An integer value of type AT_MODE that tells ImageGear what bit depth the alpha channel should have. The possible settings for this variable, which are defined in accucnst.h are: IG_ALPHA_CREATE_1 and IG_ALPHA_CREATE_8.

Description:

This function creates an alpha channel in the image **hIGear**, based on the data found in **hIBackgrnd**. The height and width of **hIBackgrnd** must not be less than the height and width of **hIGear**. If there is already an alpha channel in the image, it will be replaced. Here is the formula by which the alpha channel data is calculated (where **I2** stands for second image):

A = (I2 - Back) / (abs(I2 - back) - back);

If **hIBackgrnd** is a 1-bit image, you should set **nCreateMode** to **IG_ALPHA_CREATE_1**. When **hIGear** is displayed, this data will act as overlay data, where the 2 possible values for each bit of overlay data will determine whether the pixel is displayed or is made transparent, so that whatever is in the background will show through.

If **hIBackgrnd** is an 8-bit image, you should set **nCreateMode** to **IG_ALPHA_CREATE_8**. This will add 8 bits (with 256 possible values) of alpha data for each pixel of hIGear.

NOTE: The image must support the storage of alpha data. Targa (*.tga) is an example of one that does. In a 24-bit Targa image, each pixel is stored to 32 bits, where the extra 8 bits may be used for alpha data.

NOTE: See also **IG_IP_blend_with_alpha()** function.

Bits Per Pixel:

1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HIGEAR  hIGear,  /* HIGEAR handles of images */
HIGEAR  hIBackgrnd;
AT_ERRCOUNT    nErrcount;  /* Tally of ImageGear errors on the
stack*/
```

```
nErrcount = IG_load_file ("Picture1.tga", &hIGear);
nErrcount = IG_load_file( "Picture2".bmp", &hIBackgrnd);
nErrcount = IG_image_create_alpha( hIGear, hIBackgrnd,
IG_ALPHA_CREATE_8);
```

IG_image_create_DIB

Group: Image Utility

NOTE: Please, use the new upgraded function **IG_image_create_DIB_ex**().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_DIB(
       AT_DIMENSION nWidth,
       AT_DIMENSION nHeight,
       UINT   nBitsPerPixel,
       LPAT_DIB   lpDIB,
       LPHIGEAR   lphIGear
       );
```

Arguments:

nWidth	Set to the width that the image will be, in pixels. If the DIB already exists (lpDIB <> NULL), this value will be ignored.
nHeight	Set to the height that the image will be (number of rows). If the DIB already exists (lpDIB <> NULL), this value will be ignored
nBitsPerPixel	Set to the bit depth of the new DIB. If the DIB already exists (lpDIB <> NULL), this value will be ignored .
lpDIB	Far pointer to a DIB to copy, or NULL if creating an empty DIB. See the tip below. If this parameter is not NULL, it must be a valid pointer to the uncompressed bitmap. For example, the biCompression field of lpDIB can be either: IG_BI_RGB = 0 or IG_BI_GRAYSCALE = 503
LphIGear	A far pointer that returns a HIGEAR handle for the DIB just created

Description:

NOTE: The functionality of this API call has been upgraded and supported by the new function **IG_image_create_DIB_ex**(). The reason that this new function has been created is that the old function cannot support 16-bit DIBs. In the interest of

backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

This function creates a new DIB and returns you its HIGEAR handle. If the FAR pointer lpDIB = NULL, an empty DIB is created using arguments nWidth, nHeight, and nBitsPerPixel. If lpDIB is not NULL, it should be a FAR pointer to an existing DIB which is to be copied. The DIB to be copied need not have a HIGEAR handle associated with it. The width, height, and Bits Per Pixel will be copied from the existing DIB; arguments nWidth, nHeight, and nBitsPerPixel will be ignored.

If you have an existing DIB which you simply want to give a HIGEAR handle to, use function **IG_image_DIB_import**(), which does not make a copy of the DIB.

If the lpDIB parameter is not NULL, then it must be a valid pointer to the uncompressed bitmap, that is the biCompression field of the lpDIB structure can be either IG_BI_RGB = 0 or IG_BI_GRAYSCALE = 503.

NOTE: If you set lpDIB to NULL in order to create an empty DIB, the DIB palette will not be initialized. You will have to initialize it yourself. If you do not, the image will be displayed as all black.
Each raster in the DIB data must be padded to 32 bits. ImageGear does not support a top-down DIB (where biHeight is negative).

Bits Per Pixel:

1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
(See also the example for function IG_dspl_DDB_import).
HIGEAR       hIGearNew;      /* Will be handle of new empty DIB
*/
AT_DIMENSION  nWid, nHi;      /* Dimensions for empty DIB  */
UINT      Bpp;             /* Bits per pixel for empty DIB */
AT_ERRCOUNT  nErrCount;      /* Count of errors put on stack */
HIGEAR       hIGearCopy;     /* Will be handle of new copied DIB
*/
char FAR     *lpExistingDIB; /* Holds address of an existing
DIB */
/* Create an empty 500 x 300 x 16 bits per pixel DIB:   */
nWid  =  500;  nHi = 300;   /* Create a 500 pixel x 300 row DIB
*/
nBpp  =  16;                /* 16 bits per pixel:  */
nErrCount =  IG_image_create_DIB (nWid, nHi, nBpp, NULL,
&hIGearNew);
```

```
if ( nErrs ) { ...}        /* Process any errors */
 ...
/* Copy DIB at *lpExistingDIB, creating HIGEAR image hIGearCopy:
*/
nErrCount =  IG_image_create_DIB (0, 0, 0,  (LPAT_DIB)
lpExistingDIB, &hIGearCopy);
if ( nErrs ) { ...}        /* Process any errors */
```

Sample: IMGUTIL

IG_image_create_DIB_ex

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_DIB_ex(
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   UINT   nBitsPerPixel,
   AT_LMODE lCompression,
   LPAT_DIB lpDIB,
   LPHIGEAR lphIGear
   );
```

Arguments:

nWidth	Set to the width that the image will be, in pixels. If the DIB already exists (lpDIB <> NULL), this value will be ignored.
nHeight	Set to the height that the image will be (number of rows). If the DIB already exists (lpDIB <> NULL), this value will be ignored
nBitsPerPixel	Set to the bit depth of the new DIB. If the DIB already exists (lpDIB <> NULL), this value will be ignored .
lCompression	Set to the type of pixel storage format you would like used in the new DIB. Currently, there are three options: IG_BI_RGB - for standard DIB pixel storage. IG_BI_GRAYSCALE - for 16-bitgrayscale DIB pixel storage. IG_BI_CMYK - for 32-bit CMYK DIB pixel storage. **This variable is named lCompression because it is used to set up the biCompression field of the DIB header.
lpDIB	Far pointer to a DIB to copy, or NULL if creating an empty DIB. See the tip below. If this parameter is not NULL, it must be a valid pointer to the uncompressed bitmap. For example, the biCompression field of lpDIB can be either: IG_BI_RGB = 0 or IG_BI_GRAYSCALE = 503
LphIGear	A far pointer that returns a HIGEAR handle for the DIB just created.

Description:

This function creates a new DIB and returns you its **HIGEAR** handle. If the **FAR** pointer **lpDIB** = NULL, an empty DIB is created using arguments **nWidth**, **nHeight**, and **nBitsPerPixel**. If **lpDIB** is not NULL, it should be a **FAR** pointer to an existing DIB which is to be copied. The DIB to be copied need not have a **HIGEAR** handle associated with it. The width, height, and Bits Per Pixel will be copied from the existing DIB; arguments **nWidth**, **nHeight**, and **nBitsPerPixel** will be ignored.

If the lpDIB parameter is not NULL, then it must be a valid pointer to the uncompressed bitmap, that is the biCompression field of the lpDIB structure can be either IG_BI_RGB **=** 0 or IG_BI_GRAYSCALE **=** 503.

NOTE: If you set **lpDIB** to **NULL** in order to create an empty DIB, the DIB palette will not be initialized. You will have to initialize it yourself. If you do not, the image will be displayed as all black.

Each raster in the DIB data must be padded to 32 bits. ImageGear does not support a top-down DIB (where biHeight is negative).

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32 (CMYK)

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGearNew = NULL; // Will be handle of new empty DIB
AT_DIMENSION nWidth=0, nHight=0; // Dimensions for empty DIB
UINT nBpp = 0; // Bits per pixel for empty DIB
AT_LMODE nCompression = IG_COMPRESSION_NONE;
AT_ERRCOUNT nErrCount = 0; // Count of errors put on stack
HIGEAR hIGearCopy = NULL; // Will be handle of new copied DIB
char FAR *lpExistingDIB = NULL; // Holds address of an existing
DIB

// Create an empty 500 x 300 x 16 bits per pixel DIB
nWidth = 500; nHight = 300; nBpp = 16;
nErrCount = IG_image_create_DIB_ex (nWidth, nHight, nBpp,
nCompression, NULL, &hIGearNew);
if( nErrCount ) //Process any errors

// Copy DIB at *lpExistingDIB, creating HIGEAR image hIGearCopy
nErrCount = IG_image_create_DIB_ex (0, 0, 0, 0, (LPAT_DIB)
lpExistingDIB, &hIGearCopy);
if( nErrCount ) //Process any errors
```

IG_image_create_empty

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_empty(
HIGDIBINFO hDIB,
HIGEAR *lphIGear
   );
```

Arguments:

hDIB	DIB info object with parameters used to create image
lphIGear	Returned HIGEAR handle of created image

Description:

This function creates a new empty image. The resulting image does not have pixel data allocated.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
/* Create a new image with the same parameters as
   an existing image, but with no pixel data allocated */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGDIBINFO hDIB;        /* DIB info handle of image */
HIGEAR hImageCreated;   /* Handle of created image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIB);
nErrcount = IG_image_create_empty(hDIB, &hImageCreated);
/* ... */
IG_DIB_info_delete(hDIB);
IG_image_delete(hImageCreated);
```

IG_image_delete

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_delete (HIGEAR hIGear );
```

Arguments:

hIGear	HIGEAR handle of image to delete

Description:

This function removes the **HIGEAR** handle of **hIGear**, and frees the memory of the associated **HIGEAR** structure. This function also frees the memory associated with the image's DIB, if ImageGear allocated the DIB memory. If ImageGear did not allocate the DIB memory, the DIB continues to exist and it is the responsibility of your application to free this memory when done with it

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /*ImageGear handle of an existing image
*/
```

```
AT_ERRCOUNT nErrcount;   /*Count of errs on stack upon return
from func */
/* Invalidate handle, free HIGEAR mem (and DIB mem if alloc'd by
ImageGear)*/
nErrcount = IG_image_delete ( hIGear );
 …
/* hIGear is now not associated with any image  */
/* hIGear can now be used to hold handle associated with new
image loaded here: */
nErrcount = IG_load_file ( "picture.bmp" , &hIGear );
```

IG_image_DIB_export

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_export(
const HIGEAR                  hIGear,
AT_VOID                       *lpBuffer,
AT_INT                        BufferSize,
const AT_DIB_EXPORT_OPTIONS   *lpOptions
   );
```

Arguments:

hIGear	HIGEAR handle of image to export DIB from
lpBuffer	Memory buffer where DIB will be exported
BufferSize	Size of memory buffer - use IG_image_DIB_export_size_calc to determine what to use for this
lpOptions	Export options: Format: IG_DIB_EXPORT_FORMAT_WINDOWS - Export DIB in standard Windows DIB format. 9-16bpp grayscale images cannot be exported in this format. CMYK images will be exported as24bpp RGB. IG_DIB_EXPORT_FORMAT_IG_LEGACY - Export DIB in ImageGear legacy format. In this format, DIB compression can be IG_BI_GRAYSCALE for 9-16bpp grayscale, in which case the DIB bit depth will be the actual bit depth from 9 to 16. Also, DIB compression can be IG_BI_CMYK for CMYK images, in which case the DIB bit depth will be 32. UseAlpha: If the image contains an alpha channel, and this is set to TRUE, the alpha channel will be blended into the color channel(s) during export. imgRect: A rectangle that specifies an area of the image to be exported. If 0 is specified for the width or height, the image dimensions will be used instead.

Description:

This function exports the contents of `hIGear` into a buffer, provided by the application, using Windows DIB or ImageGear `AT_DIB` format. `hIGear` remains valid after calling this function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
HIGEAR hImported;       /* Handle of image with imported DIB */
LPAT_DIB lpDIBBuffer;   /* Buffer to store exported DIB */
AT_INT DIBSize;         /* Size of exported DIB in bytes */
AT_DIB_EXPORT_OPTIONS Options; /* DIB export options */
LPAT_BYTE lpBits;       /* Pointer to image bits */
AT_DWORD i = 0;
```

```
/* Load a 24-bit RGB image and export it as a DIB */
nErrcount = IG_load_file("test.jpg", &hImage);
Options.Format = IG_DIB_EXPORT_FORMAT_IG_LEGACY;
Options.UseAlpha = FALSE;
memset(&Options.imgRect, 0, sizeof(AT_RECTANGLE));
nErrcount = IG_image_DIB_export_size_calc(hImage,
    &DIBSize, &Options);
lpDIBBuffer = (LPAT_DIB) malloc(DIBSize);
nErrcount = IG_image_DIB_export(hImage, lpDIBBuffer,
    DIBSize, &Options);
/* Add some random noise to the DIB */
lpBits = (LPAT_BYTE) lpDIBBuffer + sizeof(AT_DIB);
while (i < lpDIBBuffer->biSizeImage - 3)
{
    lpBits[i] = 0;
    lpBits[i + 1] = 0;
    lpBits[i + 2] = 0;
    i += (rand() % 7) + 3;
}
/* Import the DIB and save it as a BMP file */
nErrcount = IG_image_DIB_import(lpDIBBuffer, &hImported);
nErrcount = IG_save_file(hImported, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImported);
IG_image_delete(hImage);
free(lpDIBBuffer);
```

IG_image_DIB_export_size_calc

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_export_size_calc(
const HIGEAR                    hIGear,
AT_INT                          *lpDIBSize,
const AT_DIB_EXPORT_OPTIONS     *lpOptions
   );
```

Arguments:

hIGear	HIGEAR handle of image to calculate exported DIB size for
lpDIBSize	Pointer to where exported DIB size will be stored
lpOptions	Export options

Description:

This function calculates the size of the exported DIB for an image. In order to export an image's DIB, you will need to pass a memory buffer with at least this many bytes allocated to IG_image_DIB_export.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
HIGEAR hImported;       /* Handle of image with imported DIB */
LPAT_DIB lpDIBBuffer;   /* Buffer to store exported DIB */
AT_INT DIBSize;         /* Size of exported DIB in bytes */
AT_DIB_EXPORT_OPTIONS Options; /* DIB export options */
LPAT_BYTE lpBits;       /* Pointer to image bits */
AT_DWORD i = 0;
/* Load a 24-bit RGB image and export it as a DIB */
nErrcount = IG_load_file("test.jpg", &hImage);
Options.Format = IG_DIB_EXPORT_FORMAT_IG_LEGACY;
Options.UseAlpha = FALSE;
memset(&Options.imgRect, 0, sizeof(AT_RECTANGLE));
nErrcount = IG_image_DIB_export_size_calc(hImage,
    &DIBSize, &Options);
lpDIBBuffer = (LPAT_DIB) malloc(DIBSize);
nErrcount = IG_image_DIB_export(hImage, lpDIBBuffer,
    DIBSize, &Options);
/* Add some random noise to the DIB */
lpBits = (LPAT_BYTE) lpDIBBuffer + sizeof(AT_DIB);
while (i < lpDIBBuffer->biSizeImage - 3)
{
    lpBits[i] = 0;
    lpBits[i + 1] = 0;
    lpBits[i + 2] = 0;
    i += (rand() % 7) + 3;
}
/* Import the DIB and save it as a BMP file */
nErrcount = IG_image_DIB_import(lpDIBBuffer, &hImported);
nErrcount = IG_save_file(hImported, "test.bmp",
```

```
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImported);
IG_image_delete(hImage);
free(lpDIBBuffer);
```

IG_image_DIB_import

Group: Format Filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_import(
const AT_DIB*    lpDIB,
LPHIGEAR         lphIGear
    );
```

Arguments:

lpDIB	DIB to be imported
lphIGear	HIGEAR handle of image created from imported DIB

Description:

This function creates a new `HIGEAR` image from a Windows DIB stored in memory. This function copies pixels to the new `HIGEAR`. Application continues to own `lpDIB` after calling this function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
HIGEAR hImported;       /* Handle of image with imported DIB */
LPAT_DIB lpDIBBuffer;   /* Buffer to store exported DIB */
AT_INT DIBSize;         /* Size of exported DIB in bytes */
AT_DIB_EXPORT_OPTIONS Options; /* DIB export options */
LPAT_BYTE lpBits;       /* Pointer to image bits */
AT_DWORD i = 0;
/* Load a 24-bit RGB image and export it as a DIB */
nErrcount = IG_load_file("test.jpg", &hImage);
Options.Format = IG_DIB_EXPORT_FORMAT_IG_LEGACY;
Options.UseAlpha = FALSE;
memset(&Options.imgRect, 0, sizeof(AT_RECTANGLE));
nErrcount = IG_image_DIB_export_size_calc(hImage,
```

```
    &DIBSize, &Options);
lpDIBBuffer = (LPAT_DIB) malloc(DIBSize);
nErrcount = IG_image_DIB_export(hImage, lpDIBBuffer,
    DIBSize, &Options);
/* Add some random noise to the DIB */
lpBits = (LPAT_BYTE) lpDIBBuffer + sizeof(AT_DIB);
while (i < lpDIBBuffer->biSizeImage - 3)
{
    lpBits[i] = 0;
    lpBits[i + 1] = 0;
    lpBits[i + 2] = 0;
    i += (rand() % 7) + 3;
}
/* Import the DIB and save it as a BMP file */
nErrcount = IG_image_DIB_import(lpDIBBuffer, &hImported);
nErrcount = IG_save_file(hImported, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImported);
IG_image_delete(hImage);
free(lpDIBBuffer);
```

IG_image_DIB_info_get

Group: DIB Info Services

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_info_get(
HIGEAR hIGear,
HIGDIBINFO *lphDIB
   );
```

Arguments:

hIGear	HIGEAR handle of image to create DIB info object from
lphDIB	Returned DIB info object handle

Description:

This function returns the handle of a new DIB info object containing information about the given image. The application must delete it after it is done using it by calling `IG_DIB_info_delete()`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT nChannels;       /* Returned number of channels */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nChannels = IG_DIB_channel_count_get(hDIBInfo);
IG_DIB_info_delete(hDIBInfo);
```

IG_image_DIB_palette_pntr_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_palette_pntr_get (HIGEAR
hIGear, LPAT_RGBQUAD FAR *lpRGBQ, LPUINT lpEntries );
```

Arguments:

hIGear	HIGEAR handle of image
lpRGBQ	Far pointer to a variable of type LPAT_RGBQUAD, in which this function will store the address of the start of the image's DIB palette
lpEntries	Far pointer to a variable in which will be returned the number of entries in the palette

Description:

This function is called to obtain the address of a **HIGEAR** image's DIB palette. Since a DIB palette consists of a list of **AT_RGBQUAD** structs, this function returns the address of the palette's first **AT_RGBQUAD** struct.

NOTE: See the section ImageGear Data Types and Structures, for a description of the **AT_RGBQUAD** struct.

When referencing a DIB palette, remember that each DIB palette entry contains 4 bytes, not 3, and that the order of the bytes is: **Blue**, **Green**, **Red**, **Unused** (not Red, Green, Blue).

NOTE: Directly altering a DIB's header, or its palette size or bitmap size, without using the HIGEAR can cause problems. ImageGear's internal HIGEAR structure keeps track of the DIB and other internal items. These fields can not be updated correctly if you alter the DIB without using the HIGEAR.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See the example for function **IG_palette_save()**.

IG_image_DIB_raster_pntr_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_raster_pntr_get(
HIGEAR hIGear,
AT_PIXPOS row,
AT_VOID **lplpRaster
      );
```

Arguments:

hIGear	HIGEAR handle of image
row	Row in image to retrieve raster pointer for
lplpRaster	Returned pointer to the 1st pixel in the raster

Description:

This function gets a pointer to the beginning of pixel data for a raster in the given image.

As long as the image is not 1bpp B/W, you can use this pointer to access other rasters in the image. So you could get a pointer to the first raster and use that pointer to access the entire image.

Rasters are stored from top to bottom, are DWORD-padded, use 8, 16, or 32 bits to store channel values, and have alpha/extra channels included in-line with the color channels. For example, an RGB image with an alpha channel would be stored as: RGBA RGBA RGBA ...

If the image is 1bpp b/w, it is stored in a compressed run ends format. See **IG_runs_row_get()**/**IG_runs_row_set()** for a description of this format. It is recommended to use these functions to access run ends data, but you can use IG_image_DIB_raster_pntr_get() with the following restrictions:

1. You can only read data. It is not safe to write data.
2. You can only access the raster you've retrieved a pointer to.

So you have to call IG_image_DIB_raster_pntr_get() for each raster you want to access.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
LPAT_VOID lpRst;        /* Pointer to a raster */
nErrcount = IG_image_DIB_raster_pntr_get(hImage, 0, &lpRst);
/* lpRst now points to first pixel in topmost raster */
```

IG_image_dimensions_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_dimensions_get (HIGEAR hIGear,
LPAT_DIMENSION lpWidth, LPAT_DIMENSION lpHeight, LPUINT
lpBitsPerPixel );
```

Arguments:

hIGear	ImageGear handle of image
lpWidth	Far pointer to where to return width of image (number of pixels per row)
lpHeight	Far pointer to where to return height (number of rows in image)
lpBitsPerPixel	Far pointer to where to return depth of image (number of Bits Per Pixel)

Description:

This function will return to you the width, height, and number of Bits Per Pixel, from the DIB of the image indicated by handle **hIGear**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;           /* Handle of image */
AT_DIMENSION nWidth, nHeight;  /* Will hold returned width and
height  */
```

```
UINT   nBpp           /* Will hold returned bits per pixel    */
AT_ERRCOUNT nErrcount      /* Will hold returned error count
*/

nErrcount = IG_image_dimensions_get ( hIGear, &nWidth, &nHeight,
&nBpp );
```

IG_image_duplicate

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_duplicate (HIGEAR hIGear, LPHIGEAR
lphIGear);
```

Arguments:

hIGear	HIGEAR handle to the current image to be duplicated
lphIGear	A far pointer in which the handle of the new, duplicate image is returned.

Description:

This function creates an exact duplicate of the current **HIGEAR** image, and returns the handle to the new image to you in **lphIGear**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR  hIGear;/* HIGEAR handle of image  */
HIGEAR lphIGear;  /* HIGEAR handle to new duplicate image   */
IG_image_duplicate(hIGear, &lphIGear);
```

IG_image_grayscale_LUT_copy_get

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_grayscale_LUT_copy_get(
   HIGEAR hIGear,
   HIGLUT *lpLUT
);
```

Arguments:

hIGear	ImageGear handle.
lpLUT	New LUT handle.

Description:

This function returns a copy of image grayscale LUT, if it exists.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

None

IG_image_grayscale_LUT_exists

Declaration:

```
AT_BOOL ACCUAPI IG_image_grayscale_LUT_exists(
HIGEAR hIGear
);
```

Arguments:

hIGear	HIGEAR handle.

Description:

This function checks whether `HIGEAR` has a grayscale LUT attached.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

None

IG_image_grayscale_LUT_update_from

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_grayscale_LUT_update_from(
    HIGEAR hIGear,
    HIGLUT lut
);
```

Arguments:

hIGear	HIGEAR handle.
lut	LUT handle.

Description:

This function updates (creates if not present) a grayscale LUT for the image.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

IG_image_is_gray

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_is_gray ( HIGEAR hIGear, LPBOOL
lpIsImageGray );
```

Arguments:

hIGear	HIGEAR handle of image
lpIsImageGray	Far pointer to a BOOL variable in which will be returned TRUE if the image is a gray scale image

Description:

This function is called to determine if an image is a grayscale image. To be considered grayscale by this function, the image must be indexed (has a palette), grayscale (has no palette), or 24 bit RGB.

An indexed image is grayscale if each palette entry is equal to the palette index offset for red, green, and blue. The shades of gray in the palette must be sorted from black to white or white to black. If the palette contains only shades of gray but is not sorted, this function will return **FALSE**.

For a 24-bit image all pixels in the image must have pixels where red, green, and blue are all equal to one another. If this condition is failed for any pixel in the image, this function will return **FALSE**.

NOTE: **FALSE** is returned for all 1-bit images, even thought the palette may contain only 2 shades of gray.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of image   */
BOOL    bItsGray;   /* Will be set = TRUE if gray-scale */
IG_image_is_gray ( hIGear, &bItsGray );
```

IG_image_is_signed_set

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_is_signed_set (
   HIGEAR hIGear,
   BOOL bSigned
   );
```

Arguments:

hIGear	HIGEAR handle of image
bSigned	Indicates whether or not grayscale image should be treated as signed.

Description:

This function is available for all versions of ImageGear. It is used on gray-scale images (8-bit gray level and 9-16-bit gray level). When ImageGear loaded the image, it attempted to determine if the pixel representations were signed or unsigned. If it could not determine, then it assumed unsigned. Use this function to force the image-attribute to either TRUE or FALSE.

NOTE: Note that this function DOES NOT modify the pixel-values themselves - it only changes the way in which ImageGear will display them. Call **IG_dspl_image_draw()** after IG_image_is_signed_set**()** to repaint the image.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_image_is_valid

Group: Image Utility

Declaration:

```
BOOL ACCUAPI IG_image_is_valid ( HIGEAR hIGear );
```

Arguments:

hIGear	HIGEAR handle of image

Description:

This function is called to determine if the **HIGEAR** variable **hIGear** contains a handle of a valid ImageGear image. Note that the return-type of the function is **BOOL**, not **AT_ERRCOUNT**. TRUE is returned if the handle is valid and may be used as the **HIGEAR** argument in calls to other ImageGear functions.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

This function returns TRUE if a **hIGear** contains a valid handle, FALSE otherwise

Example:

```
HIGEAR  hIGear;     /* Contains HIGEAR handle of image   */
if ( IG_image_is_valid ( hIGear ) )
   { IG_save_file ( hIGear, “picture.bmp”, IG_SAVE_BMP_UNCOMP
); }
```

IG_image_orientation_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_orientation_get(HIGEAR hIGear,
LPAT_MODE lpOrientation);
```

Arguments:

hIGear	HIGEAR handle of image
lpOrientation	A constant of type AT_MODE that will return the current orientation of the HIGEAR image

Description:

This function tells you what orientation the HIGEAR image was saved with. The two most-often-used orientations are “Portrait” and “Landscape.” However, in order to provide complete support for the TIFF file format, which defines eight image orientations, ImageGear interprets eight orientations. **lpOrientation** will return the orientation mode used, as an **IG_ORIENT_**… constant. For an image with a “Portrait” orientation, this value would normally be **IG_ORIENT_TOP_LEFT.** For an image with “Landscape” orientation, this value would be either **IG_ORIENT_RIGHT_TOP** or **IG_ORIENT_LEFT_BOTTOM**. The full list of constants is defined in accucnst.h.

Notice that the **IG_ORIENT** constants contain indicators of two directions. For **IG_ORIENT_TOP_LEFT**, the first direction is “**TOP**.” This specifies the placement of row 0 of the image. The second direction is “**LEFT**”, and this specifies the placement of column 0. Thus, **IG_ORIENT_TOP_LEFT** specifies that row 0 of the image stored in the file should be displayed at the top and column 0 should be displayed at the left. This is the normal orientation of most images. On the other hand, **IG_ORIENT_LEFT_BOTTOM** specifies that row 0 should be displayed at the left and column 0 at the bottom. For this to be true the image would have to be rotated 90° counter clockwise.

NOTE: The orientation setting of an image is for informational purposes only. This information is stored in the header structure of those file formats that support the storage of orientation information. The orientation setting tells how the image was intended to be displayed. In the last example above, the bitmap image is not necessarily stored “sideways.” When you find that it is intended to be displayed sideways, you could call **IG_dspl_orientation_get ()** to display it in its intended orientation, or call **IG_IP_rotate_multiple_90()** to rearrange the actual bitmap data so that the image is actually stored sideways. See example code below.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_ERRCOUNT nErrcount;    /* Tally of any ImageGear errors that
occur */
HIGEAR hIGear;   /* Contains HIGEAR handle of image  */
 …
nErrcount = IG_image_orientation_get(hIGear,
   &nOrientation);
if (nOrientation == IG_ORIENT_LEFT_BOTTOM)
{
/* if the image is meant to be displayed with row 0 at left and
col 0 at bottom, set the display angle to rotate 90°
counterclockwise */
   nErrcount = IG_dspl_orientation_set(hIGear, IG_GRP_DEFAULT,
IG_DSPL_ORIENT_LEFT_BOTTOM);
   }
```

IG_image_orientation_set

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_orientation_set(HIGEAR hIGear,
AT_MODE nOrientation);
```

Arguments:

hIGear	HIGEAR handle of image
nOrientation	A constant of type AT_MODE that sets the orientation of the HIGEAR image

Description:

This function sets the orientation of the currently loaded HIGEAR image. The two most-often-used orientations are “Portrait” and “Landscape.” However, in order to provide complete support for the TIFF file format, ImageGear interprets eight orientations. **nOrientation** is set to an **IG_ORIENT**_… constant. To set the orientation to “Portrait” you would normally use **IG_ORIENT_TOP_LEFT.** To set the orientation to “Landscape”, use either **IG_ORIENT_RIGHT_TOP** or **IG_ORIENT_LEFT_BOTTOM**. The full list of constants is defined in accucnst.h.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See example under **IG_image_orientation_get()** function.

IG_image_resolution_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_resolution_get(
    HIGEAR hIGear,
    LPLONG lpXResNumerator,
    LPLONG lpXResDenominator,
    LPLONG lpYResNumerator,
    LPLONG lpYResDenominator,
    LPAT_MODE lpnUnits
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpXResNumerator	A far pointer in which is returned the setting the numerator of the x resolution of an image
lpXResDenominator	A far pointer in which is returned the setting for the denominator of the y resolution of an image
lpYResNumerator	A far pointer in which is returned the setting for the numerator of the y resolution of an image
lpYResDenominator	A far pointer in which is returned the setting for the denominator of the y resolution of an image
lpnUnits	A far pointer to an AT_MODE variable, in which is returned the units of resolution for the image. See description for **IG_image_resolution_set()**.

Description:

This function is used to retrieve the current resolution settings of the **HIGEAR** image. It returns the values and units of resolution stored in the **HIGEAR**. To set these values, use **IG_image_resolution_set()**. See that function's description for more details about **DIB** and **HIGEAR** resolution settings.

The image resolution in the **HIGEAR** is stored as a pair of 2 ratio: one for horizontal, or "x" resolution, and one for vertical, or "y" resolution. Each ratio is a pair of **LONG**s. This is the method used by several file formats which store rational values but need to avoid IEEE types of float and double. If the X resolution numerator is 600 and the denominator is 2, then the x resolution is 300 (600/2).

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;
LONG   XResNumerator;
LONG   XResDenominator;
LONG   YResNumerator;
LONG   YResDenominator;
AT_MODE Units;
AT_ERRCOUNT nErrcount;
/* Change the current resolution setting to INCHES */
nErrcount = IG_image_resolution_get(hIGear, &XResNumerator,
&XResDenominator, &YResNumerator, &YResDenominator, &nUnits);
if (nUnits != IG_RESOLUTION_NO_ABS)
   {
   if (nUnits == IG_RESOLUTION_METERS)
      {
```

```
            /*convert from meters to inches*/
            xResNumerator = XResNumerator * 39.37;
            xResDenominator = XResDenominator * 39.37;
            yResNumerator = YResNumerator * 39.37;
            yResDenominator = YResDenominator * 39.37;
            }
        else if
            (nUnits == IG_RESOLUTION_CENTIMETERS)
            {
                /*convert from centimeters to inches*/
                xResNumerator = XResNumerator * 0.3937;
                xResDenominator = XResDenominator * 0.3937;
                yResNumerator = YResNumerator * 0.3937;
                yResDenominator = YResDenominator * 0.3937;
            }
        }
    nUnits = IG_RESOLUTION_INCHES;
    nErrcount = IG_image_resolution_set(hIGear, xResNumerator,
    xResDenominator, yResNumerator, yResDenominator, nUnits);
```

IG_image_resolution_set

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_resolution_set(
    HIGEAR hIGear,
    LONG xResNumerator,
    LONG xResDenominator,
    LONG yResNumerator,
    LONG yResDenominator,
    AT_MODE nUnits
    );
```

Arguments:

hIGear	HIGEAR handle to the image
XResNumerator	A variable of type LONG for setting the numerator of the x resolution of an image
XResDenominator	A variable of type LONG for setting the denominator of the y resolution of an image
YResNumerator	A variable of type LONG for setting the numerator of the y resolution of an image
YResDenominator	A variable of type LONG for setting the denominator of the y resolution of an image
nUnits	A constant of type AT_MODE, that sets the units of resolution for the image. Choices include: ■ IG_RESOLUTION_NO_ABS = No Absolute Units ■ IG_RESOLUTION_METERS = Pels Per Meter ■ IG_RESOLUTION_INCHES = Dots Per Inch ■ IG_RESOLUTION_CENTIMETERS = Pixels Per cm ■ IG_RESOLUTION_10_INCHES = Pixels Per 10 inches * ■ IG_RESOLUTION_10_CENTIMETERS = Pixels Per 10 cm*

* used by MO:DCA and IOCA.

Description:

This function is used to set and store the resolution of the image referenced by **hIGear**. The resolution settings will be made to both the **HIGEAR** and the **DIB**. For the **HIGEAR**, the values are saved as numerators and denominators for better precision (details below). In addition, ImageGear will also store the units of resolution to the **HIGEAR**, in **nUnits**. This allows the flexibility of storing any one of the different methods of resolution measurement listed above. Because the **DIB** specification requires that resolution be stored as pels-per-meter, the values will be converted to pels-per-meter and stored directly to the DIB's **XPelsPerMeter** and **YPelsPerMeter** fields.

To make a conversion, you can use the appropriate conversion formula:

- pels per meter = 0.0254 * dots per inch
- dots per inch = 39.37 * pels per meter

To convert from DPI to PPM or PPM to DPI, you may call **IG_convert_DPI_to_PPM()** and **IG_convert_PPM_to_DPI()**.

When an image is saved to a file the resolution values in the **HIGEAR** are used, not the values in the DIB. They will be converted when necessary to match the units specified by the file format.

To maintain the original precision of the resolution obtained from the image file the resolutions are stored as numerator and denominator. This is the method used by several image file formats as a method of avoiding the IEEE specifications for double and float. To set the X resolution of the image to 300 DPI the numerator can be 300 and the denominator 1 (900 and 3 would also work).

The ratio of the image resolutions (X and Y) is used by ImageGear when printing the image. Changing these values does not alter the number of pixels or colors in the actual image in any way. The resolution values are for informational and printing purposes only.

NOTE: To resize or crop an image see **`IG_IP_resize()`** and **`IG_IP_crop()`**.

NOTE: The settings of the resolution directly affect the output of any of the `IG_dspl_..._print...` functions.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See the example under **`IG_image_resolution_get()`**function**.**

IG_info_get_ex

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_ex ( const LPSTR lpszFileName,
UINT nPageNumber, LPAT_MODE lpFileType, LPAT_MODE
lpCompression, HIGDIBINFO *lphDIB );
```

Arguments:

lpszFileName	Name of file about which to get information
nPageNumber	Page number to get info about if this is a multi-page (multi-image) file. Note that page numbers begin at 1, not 0. Set `nPageNumber` to 1, if this is not a multi-page file.
lpFileType	Far pointer to an `AT_MODE` variable in which file type, such as `IG_FORMAT_BMP`, will be returned
lpCompression	Far pointer to an `AT_MODE` variable in which compression type, such as `IG_COMPRESSION_RLE`, will be returned
lphDIB	Far pointer to an `HIGDIBINFO` struct to which will be returned other file information such as width, height, and Bits Per Pixel

Description:

This function obtains information about the page specified by `nPageNumber` argument without actually loading the named file. The file is opened and the header is inspected. The file must exist.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
INT nPageNumber
AT_MODE nFileType;  /* Will receive an IG_FORMAT_ constant  */
AT_MODE nCompression;/* Will receive an IG_COMPRESSION_
constant*/
HIGDIBINFO dibhInfoDIB;
AT_ERRCOUNT nErrcount;     /* Returned count of errors */
nErrcount = IG_info_get_ex ( “picture.bmp”, 2, &nFileType,
&nCompression, &dibhInfoDIB );
```

Sample: **None**

IG_info_get_FD_ex

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_FD_ex (AT_INT fd, LONG lOffset,
UINT nPage, LPAT_MODE lpFileType, LPAT_MODE lpCompression,
HIGDIBINFO *lphDIB);
```

Arguments:

fd	File Descriptor handle
lOffset	Offset within file at which the file image begins
nPage	Page number about which to get information, if a multi-page file set to 1 or greater; for a non-multi-page file set to 1.
lpFileType	Far pointer to an AT_MODE variable in which file type, such as IG_FORMAT_BMP, will be returned
lpCompression	Far pointer to an AT_MODE variable in which compression type, such as IG_COMPRESSION_RLE, will be returned
lphDIB	Far pointer to an HIGDIBINFO struct to which will be returned other file information such as width, height, and Bits Per Pixel

Description:

This function obtains information about a file that is already open and whose File Descriptor handle (**fd**) you have.

NOTE: See also **IG_info_get_ex()**.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
INT     fd      /* File Descriptor handle */
AT_MODE nFileType;  /* Will receive an IG_FORMAT_ constant */
AT_MODE nCompression; /* Will receive an IG_COMPRESSION_
constant */
HIGDIBINFO dibhInfoDIB;
AT_ERRCOUNT nErrcount;  /* Returned count of errors  */
```

```
nErrcount = IG_info_get_FD ( fd, 0, 0, &nFileType,
&nCompression, &dibhInfoDIB );
```

Sample: None

IG_info_get_mem_ex

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_mem_ex (VOID FAR32 lpImage32,
AT_UINT dwSize, UINT nPage, LPAT_MODE lpFileType, LPAT_MODE
lpCompression, HIGDIBINFO *lphDIB );
```

Arguments:

lpImage32	Far pointer to start of file image in memory
dwSize	Size of image in memory.
nPage	Page number about which to get information, if a multi page file set to 1 or greater; for a non multi page file set to 1.
lpFileType	Far pointer to an AT_MODE variable in which file type, such as IG_FORMAT_BMP, will be returned.
lpCompression	Far pointer to an AT_MODE variable in which compression type, such as IG_COMPRESSION_RLE, will be returned
lphDIB	Far pointer to an HIGDIBINFO struct to which will be returned other file information such as width, height, and Bits Per Pixel

Description:

This function obtains information about a file whose image is in memory beginning at location ***lpImage**.

NOTE: See also **IG_info_get_FD_ex()** and **IG_info_get_ex()**functions.

Bits Per Pixel: 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
VOID FAR * lpMemImage32;  /* Pointer to in-memory image of file
*/
```

```
AT_UINT dwImageSize;   /* Size of image in memory */
AT_MODE nFileType; /* Will receive an IG_FORMAT_ constant  */
AT_MODE lpCompression;  /* Will receive an IG_COMPRESSION_
constant */
HIGDIBINFO dibhInfoDIB;
AT_ERRCOUNT nErrcount; /* Returned count of errors  */

nErrcount = IG_info_get_mem ( lpMemImage32, dwImageSize, 0,
&nFileType, &nCompression, dibhInfoDIB );
```

Sample: **None**

IG_initialize

Group: Library Utility

Declaration:

```
AT_ERRCODE IG_initialize(LPVOID lpDisplay, UINT nColors, LPVOID
lpPrivateColormap);
```

Arguments:

lpDisplay	X Windows display pointer or NULL.
nColors	The number of colors to allocate for PseudoColor displays.
lpPrivateColormap	TRUE=use private colormaps.

	This argument currently isn't used but reserved for the future use. Must be set to NULL now..

Description:

This function must be called before any other ImageGear function is used.

This function initializes the ImageGear library.

The first parameter is a pointer to the X Windows display obtained from calling `XOpenDisplay()` (or equivalent). Set the display parameter to `NULL` if your application does not display any images, or use X Window System capabilities.

Use the second parameter, `nColors`, to specify the number of colors that ImageGear should allocate for PseudoColor displays. Normally, you set this number to 256.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

The use of this depends upon whether the program has a display or not. In the first example program has a display; the second does not:

```
/* Initialize ImageGear */
nErrCount = IG_initialize(display, 256, FALSE);
/* Initialize ImageGear with no display */
nErrCount = IG_initialize(NULL, 0, FALSE);
```

Sample: ImageGearDemo

See also:

`IG_close()` function.

IG_IP_alpha_create

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_alpha_create(HIGEAR hIGear,
    HIGEAR hIBackgrnd,
    AT_MODEnCreateMode,
    LPHIGEARlphAlpha
    );
```

Arguments:

hIGear	HIGEAR handle of image
hIBackgrnd	HIGEAR handle to a background image
nCreateMode	An integer value of type AT_MODE that tells ImageGear what bit depth the alpha channel should have. The possible settings for this variable, which are defined in accucnst.h are: IG_ALPHA_CREATE_1 and IG_ALPHA_CREATE_8.
lphAlpha	Far pointer to the newly created alpha channel.

Description:

This new function creates alpha channel. The difference from function **IG_image_create_alpha()** is that newly created alpha channel is not associated with any HIGEAR – it's only created and returned via last parameter lphAlpha. Other parameters have the same meaning as those of IG_image_create_alpha().

See also the section ImageGear Alpha Channel Support.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

IG_IP_area_info_get

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_area_info_get(
HIGEAR hIGear,
const LPAT_RECT lpRect,
HIGPIXEL  lpPixel,
AT_MODE   channel,
AT_MODE   info
      );
```

Arguments:

hIGear	Image to get pixel block from
lpRect	Rectangle to get info from (NULL for the whole image)
lpPixel	Block of memory to receive pixels
channel	IG_COLOR_COMP_
info	IG_DIB_AREA_INFO_

Description:

This function provides the info about the rectangle area of the image.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_IP_arithmetic

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_arithmetic (HIGEAR hIGearDest, HIGEAR
hIGearOp2, AT_MODE nOperation);
```

Arguments:

hIGearDest	HIGEAR handle of Operand 1, also the destination of the operation
hIGearOp2	HIGEAR handle of Operand 2
nOperation	An integer constant of type AT_MODE which tells which kind of arithmetic operation to perform on the intersecting pixels of the two images. See accucnst.h or the description of **IG_clipboard_paste_op_set()** for the full list of constants available.

Description:

This function performs the arithmetic or logical operation specified by **nOperation**, on each corresponding pair of pixels in **hIGearDest** and **hIGearOp2**. The result is stored in **hIGearDest**. The two images must be the same width and height. If you would like to set an image rectangle within which to perform your operations, see **IG_IP_arithmetic_rect()**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;  /* Handle of image that will change */
HIGEAR  hIGearOther;  /* Handle of image 2 */
AT_ERRCOUNT nErrcount;   /* Returned count of errors */

/* Image 1 Pixel =  Image 1 Pixel  |  Image 2 Pixel */
nErrcount = IG_IP_arithmetic ( hIGear, hIGearOther, IG_ARITH_OR
);
```

IG_IP_arithmetic_rect

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_arithmetic_rect (
   HIGEAR hIGear1,
   HIGEAR hIGear2,
   LPAT_RECTlpImageRect2,
   AT_MODEnOperation
   );
```

Arguments:

hIGear1	HIGEAR handle of image 1, which is the destination image.
hIGear2	HIGEAR handle of image 2, the images must have be of the same height and width.
lpImageRect2	A long pointer to a set of rectangle coordinates of type AT_RECT, which will define the region of image 1 to apply the arithmetic operation to. Set to NULL for the whole image.
nOperation	An integer constant of type AT_MODE which tells which kind of arithmetic operation to perform on the intersecting pixels of the two images. See accucnst.h or the description of **IG_clipboard_paste_op_set()** function for the full list of constants available.

Description:

This function performs in the same way that **IG_IP_arithmetic()** does, except that you can specify a rectangular region of **hIGear1** to perform the pixel operation on.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear1;     /* HIGEAR Handle of image 1     */
HIGEAR hIGear2;     /* HIGEAR Handle of image 2     */
AT_ERRCOUNTnErrcount;   /* Returned count of IG errors on the
stack */
AT_DIMENSION nWidth, nHeight;  /* Height and width of image */
AT_RECTrcRectOf1;   /* selected rectangle from image 1  */
UINT   nBpp;       /* bits per pixel of image        */

/* Get dimensions of hIGear1, so can use width and height values
below  */
nErrcount = IG_image_dimensions_get( hIGear1, &nWidth,
&nHeight, &nBpp );
/*Use the bottom ½ of hIGear1 for arithmetic operations
*/
rcRectOf1.left = 0;
rcRectOf1.top = nHeight/2;
rcRectOf1.right = nWidth - 1;
rcRectOf1.bottom = nHeight - 1;
nErrcount = IG_IP_arithmetic_rect( hIGear1, hIGear2,
&rcRectOf1, IG_ARITH_ADD);
```

IG_IP_blend_percent

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_blend_percent ( HIGEAR hIGearDest,
const HIGEAR hIGear2, const DOUBLE dblPctOfImage2, const AT_MODE
nColorChannel, const LPAT_RECT lpRect);
```

Arguments:

hIGearDest	HIGEAR handle of the image to be blended into
hIGear2	HIGEAR handle of image to blend in, must be same size and bit depth
dblPctOfImage2	Percent of image 2 to be in the blend (the percent of image 1 will be 100.0 minus this). 0 means all of image 1; 50 means half and half, and 100 means all of image 2. The range of values is 0.0 to 100.0
nColorChannel	A constant such as IG_COLOR_COMP_R, _G, _B, or _RGB, specifying which color(s) to blend
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. lpRect = NULL means the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non- rectangular ROI defined by the mask.

Description:

This function will blend the second image into the first. Use **lprect** to set a rectangular portion of the first image to be processed. **dblPctOfImage2** specifies the percent of each image to be used in the result. The image in **hIGearDest** is destroyed and is replaced with the resulting blend. The percentage ranges from 0 to 100. A zero results in 0% **hIGearDest** and 100% **hIGear2**. A value of 50% results in an image which is created ½ of each image. A value of 100% creates an image that is 100% **hIGear2** and 0% **hIGearDest**. The images must be the same width, height, and bit depth. **nColorChannel** lets you specify that only one color of a 24-bit image is to be blended in.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR  hIGear1;      /* Handle of image1 that will change  */
HIGEAR  hIGear2;     /* Handle of image2      */
AT_ERRCOUNT nErrcount;  /* Returned count of errors  */
AT_DIMENSION nWidth, nHeight;  /* Width & Height of Image1
*/
UINT    nBpp;     /* Bits per pixel    */
AT_RECT rcRectOf1;   /* selected rectangle from image1  */

/* Get dimensions of hIGear1, so can use width and height values
below */
nErrcount = IG_image_dimensions_get( hIGear1, &nWidth,
&nHeight, &nBpp );
/*Use the bottom ½ of Image1 */
rcRectOf1.left = 0;
rcRectOf1.top = nHeight/2;
rcRectOf1.right = nWidth - 1;
rcRectOf1.bottom = nHeight - 1;
/* If 24-bit image, blend all color channels (else blend pixels)
*/
nErrcount = IG_IP_blend_percent ( hIGear1, hIGear2,
20.0,IG_COLOR_COMP_RGB, &rcRectOf1 );
```

IG_IP_blend_with_alpha

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_blend_with_alpha ( HIGEAR hIGear1,
   UINT   nFAIndex
   HIGEAR hIGear2,
   UINT   nSAIndex
   HIGEAR hIBackgrnd,
   LPAT_RECTlpRect,
   AT_MODEnBlendMode
   );
```

Arguments:

hIGear1	HIGEAR handle of first image
nFAIndex	Specifies an index of alpha channel in hIGear1 image.
hIGear2	HIGEAR handle of second image
nSAIndex	Specifies an index of alpha channel in hIGear2 image.
hIBackgrnd	HIGEAR handle of background image
lpRect	A long pointer to a structure of type AT_RECT which tells which rectangular subregion of hIGear1 will have blending applied to it. Set to NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nBlendMode	An integer of type AT_MODE which is used to set which type of blending should be used. These are defined in accucnst.h and are shown below with their formulas.

Description:

This function blends two images, **hIGear1** and **hIGear2**, using both their pixel data, their alpha channel data (if they have any), and a background image. The changes are made to the pixel data of **hIGear1**. All three images: **hIGear1**, **hIGear2**, and **hIBackgrnd** must have the same width and height. **hIGear2** and **hIBackgrnd** may be **NULL**. **hIGear1** should not be **NULL**.

Two parameters nFAIndex and nSAIndex are used to specify index of alpha channel to be taken in blending.

The images **hIGear1** and **hIGear2** will be blended equally, and any areas that are not covered by either image will show through to the background image. If hIGear1 has a lower bit depth than **hIGear2**, it will automatically be promoted to the bit depth so that the resultant image has the higher bit depth.

Below are the possible settings for **nBlendMode**. With each setting, there is a brief description of what this blend mode will do, and also the formula that is used for calculating the new pixel values of **hIGear1**, where. Please use the following key for the formulas:

- I1 = hIGear1
- I2 = hIGear2
- Back = hIBackgrnd
- A1 = alpha channel of the first image
- A2 = alpha channel of the second image
- **IG_BLEND_OVER**: Blend first image over the second image:

I1 = (Back*(1-A1)*(1-A2)) + (I1*A1) + (I2*(1-A1)*A2)

- **IG_BLEND_IN**: Blend the first image into the second image

I1 = (Back*(1-A1)*(1-A2)) + (I1*A1*A2) + (I2*(1-A1)*A2)

- **IG_HELD_OUT**: The first image is held out by the second image.

I1 = (Back*(1-A1)*(1-A2)) + (I1*A1*(1-A2)) + (I2*A2)

- **IG_BLEND_LINEAR**: The first and second images are combined in a linear fashion.

I1 = ((Back*(1-A1)*(1-A2)) + (I1*A1) + (I2*A2)/(1+A2*A1)

Exceptions:

- if I2 ==NULL is it replaced with 0
- if A1==NULL it is replaced with 1
- if A2==NULL it is replaced with 0
- if Back==NULL it is replaced with 0.

NOTE: The values contained in the alpha channel will be mapped to a float value between 0 and 1, where 1 is equivalent to 100%. 100% means that the pixel will be fully transparent, and 0% means that the pixel will be fully opaque. All values in between (0.1 - 0.99) will represent varying degrees of transparency. If it is an 8-bit alpha channel, the value 255 will be mapped to 1, or 100%.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument so that you can blend a rectangular subregion of **hIGear1**. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

NOTE: See also **IG_image_create_alpha()**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear1, hIGear2,      /* HIGEAR handle of image */
       hIBackgrnd;
AT_ERRCOUNT nErrcount;      /* tally of IG errors on the stack
*/
AT_RECT rcRectOf1;      /* selected rectangle from image 1 */

nErrcount = IG_IP_blend_with_alpha (hIGear1, hIGear2,
hIBackgrnd, &rcRectOf1, IG_BLEND_OVER);
```

IG_IP_blend_with_LUT

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_blend_with_LUT ( HIGEAR hIGearDest,
HIGEAR hIGear2, const LPAT_LUT lpLUT_red, const LPAT_LUT
lpLUT_green, LPAT_LUT lpLUT_blue, const LPAT_RECT lpRect);
```

Arguments:

hIGearDest	HIGEAR handle of the image to be blended into
hIGear2	HIGEAR handle of image to blend in, must be same size and bit depth
lpLUT_red	Far pointer to Red channel of LUT for 24-bit image; also used as single pointer to LUT of a gray-scale image with a bit depth of 8.
lpLUT_green	Far pointer to Green channel of LUT for 24-bit image; not used for 8-bit images
lpLUT_blue	Far pointer to Blue channel of LUT of 24-bit image; not user for 8-bit images
lpRect	Rectangular portion of the image to process. Set to NULL for the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.

Description:

This function blends two images using Look-Up Tables (LUTs) to determine the strength of each pixel's contribution. The pixel values from **hIGearDest** are used as the indexes into the LUTs. Both **hIGear2** and **hIGearDest** must be the same bit depth and dimensions. The image in hIGearDest is destroyed and is replaced with the resulting blend.

For 24-bit images, each channel (R, G, or B) of the **hIGearDest** image are processed through their own LUT. For 8-bit images, the **lpLUT_red** is used and the other two are ignored (you can pass in **NULL** for these). Each LUT that is to be used must point to an 8-bit LUT that has at least enough entries to process the

images being passed in. A 24-bit blend requires 3 LUTs, each having 256 entries. (All three pointers can point to the same LUT.) An 8-bit gray level blend requires a single LUT (**lpLUT_red**) of 256 entries.

Each entry into the LUT determines the percentage of the blend on a pixel-by-pixel basis. Pixel values of the **hIGearDest** image are used as indexes into the array. The LUTs should be initialized with values from 0 to 100, where:

- 0 = 0 % hIGearDest, 100% hIGear2
- 50 = 50% of hIGearDest and 50% of hIGear2
- 100 = 100% hIGearDest and 0 % hIGear2

For 8-bit images, the value of each pixel in **hIGearDest** is used as the index into the LUT. For 24-bit images, it is the intensity (R+G+B)/3 that is used.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGearDest    /* HIGEAR handle of image 1, destination */
       hIGear2;   /* HIGEAR handle of image 2    */
AT_LUT  LUT_blend[256];   /* The LUT for blend values */
INT     pix;            /* Loop index, = pixel value  */
AT_ERRCOUNT nErrcount;     /* Returned count of errors   */
for ( pix = 0;  pix < 256;  pix++ )
   { LUTblend[pix] =  25; }    /* set Look Up Table   */
for (pix = 10; pix < 100; pix++)
   { LUTblend[pix] = 75; }
nErrcount =  IG_IP_blend_with_LUT ( hIGearDest, hIGear2,
(LPAT_LUT)&LUT_blend,
   (LPAT_LUT)&LUT_blend, (LPAT_LUT)&LUT_blend, NULL );
```

IG_IP_color_combine_ex

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_combine_ex (LPHIGEAR
lphIGear_result,
   HIGEAR hIGear1,
   HIGEAR hIGear2,
   HIGEAR hIGear3,
   HIGEAR hIGear4,
   AT_MODE color_space,
   AT_MODE dst_color_space
   );
```

Arguments:

lphIGear_result	Far pointer to an object of type HIGEAR, to receive the HIGEAR handle of the created 24- bit image
hIGear1	HIGEAR handle of input image to be treated as channel 1
hIGear2	HIGEAR handle of input image to be treated as channel 2
hIGear3	HIGEAR handle of input image to be treated as channel 3
hIGear4	HIGEAR handle of input image to be treated as channel 4
nColorSpace	A constant such as IG_COLOR_SPACE_RGB describing how the channels are to be merged
dst_color_space	tells what color space output HIGEAR should have. There are 2 possible values now: RGB and CMYK.

Description:

This function is an upgrade to **IG_IP_color_combine().** While support for **IG_IP_color_combine()** will always be maintained, it is recommended that you use this newer function. The additional benefit of this function is that it allows you to choose a color space for the destination image. For example, you can choose to store your destination image using the CMYK color scheme. As of this writing, the dst_color_space parameter can take one of the following constants as its setting: **IG_COLOR_SPACE_RGB and IG_COLOR_SPACE_CMYK**. In order to set it to CMYK, you must be sure that you first call **IG_color_space_level_set()** function with settings of **IG_COLOR_SPACE_CMYK** and **IG_FULL_SUPPORT**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

IG_IP_color_convert

Group: Image Processing

Declaration:

```
AT_ERRCOUNTACCUAPIIG_IP_color_convert(
[IN]   HIGEAR hIGear,
[IN]   AT_MODE nColorSpace
);
```

Arguments:

hIGear	The ImageGear handle of an image.
nColorSpace	The identifier of destination color space where to convert.

Description:

This function converts an image from one internal format to another depending on the parameter, `nColorSpace`. This parameter must be a color space listed in enumIGColorSpaces, which is defined in accucnst.h.

Commonly used values for `nColorSpace` are:

- `IG_COLOR_SPACE_CMYK`
- `IG_COLOR_SPACE_RGB`

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
. . .
IG_IP_color_convert( hIGear, IG_COLOR_SPACE_CMYK );
. . .
```

IG_IP_color_count_get

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_count_get(
      HIGEAR hIGear,
      LPAT_RECT lpRect,
      LPAT_INT lpCount
```

```
);
```

Arguments:

hIGear	Image whose count colors are to be counted
lpRect	Rectangle in which to count colors (NULL for the whole image)
lpCount	Pointer to AT_INT to receive number of colors

Description:

This function counts the number of unique colors in the specified rectangular area of the image.

Applicability:

ALL

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Sample:

PIX_ACC

IG_IP_color_promote

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_promote ( HIGEAR hIGear, AT_MODE
nPromoteTo);
```

Arguments:

hIGear	HIGEAR handle of image
nPromoteTo	One of these options: IG_PROMOTE_TO_4 IG_PROMOTE_TO_8 IG_PROMOTE_TO_24 IG_PROMOTE_TO_32

Description:

This function will promote an image to the common bit depths of 4, 8, 24, or 32 bits per pixel. Promotion to 4 or 8bpp produces an image in the indexed color space, 24bpp in RGB color space, and 32 in CMYK color space. The color space of the source image will be converted if necessary.

The width and height of the image are not changed. Only the number of Bits Per Pixel are increased. Colors are not changed. The same color is used for each new pixel as was used in the original.

When promoting to 24-bit, levels of gray and white may appear to be slightly colored when displayed. This is due to using a common 8-bit palette to display a 24-bit color image. The actual 24-bit pixel values are correct.

Applicability: All (with input bit depth less than output bit depth)

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of 1,4, or 8 bit image */
/* Make it a 24 bit image: */
IG_IP_color_promote ( hIGear, IG_PROMOTE_TO_24 );
```

IG_IP_color_reduce_bayer

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_bayer ( HIGEAR hIGear,
UINT nToBits, LPAT_RGBQUAD lpPalette );
```

Arguments:

hIGear	HIGEAR handle of image to reduce bit depth of
nToBits	Number of bits per pixel after reduction (4 or 1)
lpPalette	This argument is currently not used

Description:

This function reduces the image to a fewer number of Bits Per Pixel, using a Bayer dithering algorithm. The target bit depth is specified by argument **nToBits**. In general, a color image will be reduced to a fewer number of colors, and a gray-scale image will be reduced to a fewer number of shades of gray. Note that setting **nToBits** = 1 will reduce the image to monochrome or black-and-white.

The input number of Bits Per Pixel must be greater than **nToBits**, or an error will result.

NOTE: See also the section in entitled Color Reduction.

Applicability: 4bpp indexed, 8bpp indexed, 8bpp grayscale, 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of 4, 8, or 24 bit image */
/* Reduce image to 1-bit black-and-white: */
IG_IP_color_reduce_bayer ( hIGear, 1, NULL );
```

IG_IP_color_reduce_diffuse

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_diffuse ( HIGEAR hIGear,
UINT nToBits, INT level, LPAT_RGBQUAD lpPalette );
```

Arguments:

hIGear	HIGEAR handle of image to reduce bit depth of
nToBits	Number of bits per pixel after reduction, 1 or 4
nLevel	Threshold value for dithering, 0 to 255
lpPalette	This argument is currently not used

Description:

This function reduces the image to a fewer number of Bits Per Pixel. The target bit depth is specified by argument **nToBits**. In general, a color image will be reduced to a fewer number of colors, and a gray-scale image will be reduced to a fewer number of shades of gray. Note that setting **nToBits** = 1 will reduce the image to monochrome or black-and-white.

nLevel sets a threshold value for the target image. **nLevel** = 128 means that black and white in the target image will be in equal proportion. **nLevel** greater than 128 would mean more bright than dark, **nLevel** less than 128 would mean more dark than bright.

The input number of Bits Per Pixel must be greater than **nToBits**, or an error will result.

NOTE: See also section in entitled Color Reduction.

Applicability: 4bpp indexed, 8bpp indexed, 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;   /* HIGEAR handle of 4, 8, or 24 bit image */
/* Reduce image to 1-bit black-and-white: */
IG_IP_color_reduce_diffuse ( hIGear, 1, 128, NULL );
```

IG_IP_color_reduce_halftone

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_halftone ( HIGEAR
hIGear, AT_MODE nOption );
```

Arguments:

hIGear	HIGEAR handle of image
nOption	halftoning pattern - 0 for now (squares)

Description:

This function reduces a color or gray-scale image to a 1 bit per pixel image suitable for use in half-toning. The resulting image will consist of small squares of varying sizes that will give the appearance of varying shades of gray.

NOTE: See also the section in entitled Color Reduction.

Applicability: 4bpp indexed, 8bpp indexed, 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of 4, 8, or 24 bit image */
/* Reduce to a 1-bit halftone image: */
IG_IP_color_reduce_halftone ( hIGear, 0 );
```

IG_IP_color_reduce_median_cut

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_median_cut ( HIGEAR
hIGear, BOOL bFastRemap, UINT nMaxColors );
```

Arguments:

hIGear	HIGEAR handle of image
bFastRemap	Set = TRUE for reduction algorithm optimized for speed. Set = FALSE for algorithm optimized for quality
nMaxColors	The maximum number of colors (that is, maximum number of unique pixel values) you want in the resulting image. Must be 16 to 256.

Description:

This function reduces an image by dividing it into **nMaxColors** equal-sized squares. The colors of the pixels in each square will be averaged to produce a resulting color. The resulting image will contain only these colors.

NOTE: See also section in entitled Color Reduction.

Applicability: 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of image */

/* Reduce to 64 colors using median cut algorithm */
IG_IP_color_reduce_median_cut ( hIGear, FALSE, 64 );
```

IG_IP_color_reduce_octree

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_octree ( HIGEAR hIGear,
BOOL bFastRemap, UINT nMaxColors, const UINT nPaletteSize, const
LPAT_RGB lpPalette );
```

Arguments:

hIGear	HIGEAR handle of image
bFastRemap	Set = TRUE for reduction algorithm optimized for speed. Set = FALSE for algorithm optimized for quality
nMaxColors	The maximum number of colors (that is, maximum number of unique pixel values) you want in the resulting image. If you set this to > 16, the resulting image will be an 8-bit. Valid values are 8 - 256. If you pass in a value out of this range, ImageGear will set the value to either 8 or 256. See description below for details.
nPaletteSize	Number of entries in palette lpPalette that will be used by ImageGear during reduction. This should be >= nMaxColors. Setting this to 0, and setting lpPalette to NULL will have ImageGear make an optimized palette for you.
lpPalette	LONG pointer to an array of palette entries, where the number of entries = nPaletteSize. Set to NULL if you want ImageGear to make an optimized palette for you.

Description:

This function reduces a 24-bit or 8-bit image to an 8-bit or 4-bit image, having the number of colors specified by **nMaxColors**. If you set **nPaletteSize** > 0 and supply an address to **lpPalette**, ImageGear will use your palette. If you set either **nPaletteSize** = **0** or **lpPalette** to **NULL**, ImageGear will build an optimized palette for you. If you set **nMaxColors** > **16**, then an 8-bit image will always result. Setting **nMaxColors** <=**16** will result in a 4-bit image. You may not specify less than 8 colors.

nPaletteSize should be set to >= **nMaxColors**. If **nPaletteSize** is set to **0**, ImageGear will build an optimal palette and **lpPalette** will be unused. The table below demonstrates some sample cases of 8 and 24-bit images being reduced, using the setting of **nMaxColors** (middle column). The right-most column shows the number of Bits Per Pixel that the resulting image will have.

Table 3. Octree Bit Depths *In* and *Out*.

Bpp of orig. HIGEAR	# of resulting colors	Bpp reduced image
24	17-256	8
24	8-16	4
24	16	4
8	17-256	8
8	8-16	4
8	6	*

*Not possible, nMaxColors will be changed to 8

NOTE: See also the section in entitled Color Reduction.

Applicability: 8bpp indexed, 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of image   */
AT_RGB rgbPalette[64];  /* Pointer to image's palette */
AT_ERRCOUNT   nErrcount;    /* Tally of ImageGear errors on
stack*/
/* Reduce image to 64 colors using given palette */
nErrcount = IG_IP_color_reduce_octree (hIGear, FALSE, 64, 64,
rgbPalette);
/* Reduce image to 64 colors, having ImageGear build optimal
palette  */
nErrcount = IG_IP_color_reduce_octree (hIGear, FALSE, 64, 0,
NULL);
```

IG_IP_color_reduce_popularity

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_popularity ( HIGEAR
hIGear, BOOL bFastRemap, UINT nMaxColors );
```

Arguments:

hIGear	HIGEAR handle of image
bFastRemap	Set = TRUE for reduction algorithm optimized for speed. Set = FALSE for algorithm optimized for quality
nMaxColors	Number of colors to reduce to

Description:

This function reduces a 24-bit image to an 8-bit image while retaining its most popular, or prevalent, colors. Use **nMaxColors** to specify the maximum number of colors wanted in the result.

NOTE: See also the section in entitled Color Reduction.

Applicability: 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of image */
/* Reduce to 64 colors using octree algorithm   */
IG_IP_color_reduce_popularity ( hIGear, FALSE, 64 );
```

IG_IP_color_reduce_to_bitonal

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_to_bitonal(HIGEAR
hIGear, const AT_MODE nOption, const UINT nThreshold, UINT
nWeight1, UINT nWeight2, UINT nWeight3);
```

Arguments:

hIGear	HIGEAR handle of image
nOption	A constant of type AT_MODE, such as IG_REDUCE_BITONAL_AVE, that specifies what type of reduction to perform. See accucnst.h for the complete list.
nThreshold	An integer value that specifies which pixel values will be changed to black, and which pixel values will be changed to white. Pixels with values greater than nThreshold will be changed to white, and pixels that have values less than nThreshold will be changed to black.
nWeight1	These values are only used if nOption is set to IG_REDUCE_BITONAL_WEIGHTED. The range of values is 0 - 255. The default values are 255.
nWeight2	The range of values is 0 - 255. The default values are 255.
nWeight3	The range of values is 0 - 255. The default values are 255.

Description:

This function reduces the **hIGear** image from 24, 8, or 4 bpp to 1 bpp. Set **nOption** to specify how to get the threshold value. If you set nOption to IG_REDUCE_BITONAL_WEIGHTED, you may also set the values of nWeight1, nWeight2, and nWeight3. These “weights” are used to determine how much influence the values of the red, green, or blue pixels will have on the reduction. For example, if nWeight1 (red) **= 255**, nWeight2 (green) **= 0**, **nWeight3** (blue) **= 0**, the whole reduction will depend on the value of the red pixels. The green and blue pixel values will have “no weight.”

IG_REDUCE_BITONAL_GRAYSCALE gives the most weight to the value of green. This optimizes for the perception of the human eye, in which blue is the hardest color to see, and therefore requires the least weight.

IG_REDUCE_BITONAL_AVE gives equal weight to all three pixel values.

Here are the formulas used by the three different reduction methods:

- IG_REDUCE_BITONAL_GRAYSCALE: value = (red*77 + green*151 + blue*28)/256;
- IG_REDUCE_BITONAL_AVE: value = (red + green + blue)/3;
- IG_REDUCE_BITONAL_WEIGHTED: value = (red*w1 + green*w2 + blue*w3)/(w1 + w2 + w3);

Use the **nThreshold** argument to set the threshold value for converting pixels to black or white. If the value, as calculated by one of the above reduction methods, is less than **nThreshold**, the pixels will be set to black; if it is greater or equal to **nThreshold**, the pixels will be set to white.

NOTE: See also the section in entitled Color Reduction.

Applicability: 4bpp indexed, 8bpp indexed, 8bpp grayscale, 24bpp RGB

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear,      /* HIGEAR handle of input image   */
AT_ERRCOUNT nErrcount;   /* Returned count of errors  */
nErrcount = IG_IP_color_reduce_to_bitonal ( hIGear,
IG_REDUCE_BITONAL_AVE, 100, 0, 0, 0 );
```

IG_IP_color_separate

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_separate (HIGEAR hIGearOrig,
LPHIGEAR lphIGear1,LPHIGEAR lphIGear2, LPHIGEAR
lphIGear3,LPHIGEAR lphIGear4, AT_MODE nColorSpace );
```

Arguments:

hIGearOrig	HIGEAR handle of the original image to be separated.
lphIGear1	Far pointer to an object of type HIGEAR to receive the handle of the separated channel 1 image.
lphIGear2	Far pointer to receive handle of channel 2 image.
lphIGear3	Far pointer to receive handle of channel 3 image.
lphIGear4	Far pointer to receive handle of channel 4 image.
nColorSpace	A constant such as IG_COLOR_SPACE_RGB or IG_COLOR_SPACE_CMYK describing how the individual channels are to be separated or extracted.

Description:

This function is the reverse of **IG_IP_color_combine_ex()**. See the description of that function. Each of the output images created by this function (**lphIGear1**, etc.) will be gray-scale. That is, each will have a gray-scale palette. The pixel values will be those obtained from the input image.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear,    /* HIGEAR handle of input image  */
        hIGearRed, /* Handle of Red output image,   */
        hIGearGreen, hIGearBlue,   /* Green, Blue output images
*/
        hIGearNULL;    /* (Not used when IG_COLOR_SPACE_RGB)
*/
AT_ERRCOUNT nErrcount;  /* Returned count of errors */

nErrcount = IG_IP_color_separate ( hIGear, &hIGearRed,
&hIGearGreen, &hIGearBlue, &hIGearNull, IG_COLOR_SPACE_RGB );
```

IG_IP_contrast_adjust

Group: Image Processing

NOTE: See also **IG_IP_contrast_adjust_ex**.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_adjust ( HIGEAR hIGear,
LPAT_RECT lpRect, AT_MODE nMethodMode, DOUBLE dblContrast,
DOUBLE dblBrightness );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. lpRect = NULL means the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to adjust by altering the pixels themselves, or the palette
dblContrast	From -255.0 to +255.0, the adjustment in contrast.
dblBrightness	From - 255.0 to +255.0, the adjustment in brightness

Description:

This function allows the brightness and contrast of the image to be adjusted. Brightness and contrast are linear controls that affect the intensity of the image pixels. These controls are similar to the Brightness and Contrast controls on a typical television set.

Contrast is a multiplier and Brightness is an additive value. A Contrast of 2.0 will cause each pixel to take on twice the intensity that it originally had while 0.5 will divide each by 2. A Brightness value of 20.0 will cause each pixel's intensity to be increased by 20 and a -20 will decrease or darken each by 20. Pixel values are clipped to the 0 to 255 range. Once clipped the data is lost and can not be regenerated. A Brightness of 0.0 and a Contrast of 1.0 will cause no change to the image. A -1.0 Contrast with a Brightness of 0.0 can be used to invert the intensity range.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: See the functions **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

When **IG_CONTRAST_PALETE** is used the **lpRect** rectangle is ignored, since the whole image is affected when the palette is changed.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of image */
IG_IP_contrast_adjust ( hIGear, NULL, IG_CONTRAST_PIXEL, 2.0, -
10.0 );
```

IG_IP_contrast_adjust_ex

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_adjust_ex(
HIGEAR      hIGear,
LPAT_RECT   lpRect,
AT_MODE     method_mode,
DOUBLE      contrast,
DOUBLE      brightness,
AT_MODE     color_channel);
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. lpRect = NULL means the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
method_mode	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to adjust by altering the pixels themselves, or the palette
contrast	From -255.0 to +255.0, the adjustment in contrast.
brightness	From - 255.0 to +255.0, the adjustment in brightness
color_channel	Specifies the color channel to adjust.

Description:

This function allows the brightness and contrast of the image to be adjusted. brightness and contrast are linear controls that affect the intensity of the image pixels. These controls are similar to the brightness and contrast controls on a typical television set.

contrast is a multiplier and brightness is an additive value. A contrast of 2.0 will cause each pixel to take on twice the intensity that it originally had while 0.5 will divide each by 2. A brightness value of 20.0 will cause each pixel's intensity to be increased by 20 and a -20 will decrease or darken each by 20. Pixel values are clipped to the 0 to 255 range. Once clipped the data is lost and can not be regenerated. A brightness of 0.0 and a contrast of 1.0 will cause no change to the image. A -1.0 contrast with a brightness of 0.0 can be used to invert the intensity range.

color_channel must be a value from enumIGColorChannels. It specifies the color channel to adjust. This channel does not need to be in the same color space as the image. For example, you can adjust the saturation of an RGB image by specifying IG_COLOR_COMP_IHS_S.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: See the functions **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** for more details.

When **IG_CONTRAST_PALETE** is used the **lpRect** rectangle is ignored, since the whole image is affected when the palette is changed.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
// Decrease the saturation of an image:
IG_IP_contrast_adjust_ex(hIGear, NULL, IG_CONTRAST_PIXEL, 1, -
128, IG_COLOR_COMP_IHS_S);
```

IG_IP_contrast_equalize

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_equalize(HIGEAR hIGear,
LPAT_RECT lpRect, AT_MODE nMethodMode );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct defining the rectangle within the image that this function is to operate on. If NULL, the entire image will be operated on. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to alter the pixels themselves (the image bitmap) or the palette

Description:

This function automatically adjusts the contrast of the image so that each range of possible intensities has about the same number of pixels in it. Just like **IG_IP_contrast_stretch()**, this will expand the intensity range of the image to fill the entire 0 to 255 range. However, unlike that function this one is non-linear.

This function is often used in x-ray images and in others where the contrast can be very small in the original. **IG_IP_contrast_equalize()** will bring out subtle changes in the contrast.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been

set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* HIGEAR handle of image */
IG_IP_contrast_equalize ( hIGear, NULL, IG_CONTRAST_PIXEL );
```

IG_IP_contrast_gamma

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_gamma ( HIGEAR hIGear,
LPAT_RECT lpRect, AT_MODE nMethodMode, DOUBLE dblGamma );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. lpRect = NULL means the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to adjust by altering the pixels themselves, or the palette
dblGamma	Greater than 0.0. Usual range: 0.75 to 3.0

Description:

Gamma is a non-linear method to adjust the contrast of a image. In this method, the amount a pixel's intensity changes depends on its original intensity. This can be used to make dark regions brighter without over saturating (clipping) the bright regions. Or, conversely, to make light regions darker without under saturating the dark regions.

Gamma was originally introduced to compensate for the non-linear nature of the phosphors used in monitors and in the original tube cameras that created images. However, it can be used as an image processing function to create better looking images that may not be mathematically color corrected.

While the gamma can be any non-zero positive value the usual range is 0.75 to 3.0. A gamma value of 1.0 does not alter the image. For typical monitors a range of 1.8 to 2.2 is usual. Values less than 1.0 cause dark pixel to become brighter. Values greater than 1.0 cause bright regions to become darker.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* HIGEAR handle of image */
IG_IP_contrast_gamma ( hIGear, NULL, IG_CONTRAST_PALETTE, 2.0 );
```

IG_IP_contrast_invert

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_invert (HIGEAR hIGear,
LPAT_RECT lpRect, AT_MODE nMethodMode );
```

Arguments:

hIGear	HIGEAR handle of image to invert contrast of
lpRect	Far pointer to an AT_RECT struct defining the rectangle within the image that this function is to operate on. If NULL, the entire image will have its contrast inverted. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to alter the pixels themselves (the image bitmap) or the palette

Description:

This function will invert every color to its complement, in the rectangular portion of the image selected by lpRect. For black-and-white images, black will become white and white will become black. For gray-scale and color images, every red, green, and blue color intensity value will be complemented: 0 will become 255 (and vice versa), 1 will become 254, and so on. Therefore, in a gray-scale image, the darkest grays will become the lightest grays, and vice versa; and in a color image, colors near green will complement to colors near magenta (the complement of green), and so on.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

If **nMethodMode** = **IG_CONTRAST_PIXEL**, the inversion is accomplished by inverting all bits of all pixels within **lpRect**: bits that are 1 become 0, and bits that are 0 become 1. If **nMethodMode** = **IG_CONTRAST_PALETTE**, the inversion is accomplished by inverting the bits in the image's palette. (The image bitmap is left unchanged.) Note that specifying **IG_CONTRAST_PALETTE** inverts the entire image, ignoring any rectangle specified. Note also that for 8-bit indexed color (8i) images, inverting the pixel will produce an error; only **IG_CONTRAST_PALETTE** makes sense.

If the image is 24-bit, only the pixel bits can be complemented (since there is no palette), so in this case `nMethodMode` is ignored.

In photographic terms, this function produces the "negative" of the image.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* Handle of image to have contrast
inverted*/
AT_RECT rcInvertRect;/* Portion of image to be inverted   */
AT_ERRCOUNT nErrcount    /* Returned number of errors        */
AT_DIMENSION nWid        /* Will hold returned width of image */
AT_DIMENSION nHi        /* Returned height of image, number of
rows */
UINT    nBpp      /* Returned number of bits per pixel */

/* The following will invert contrast on only the upper right
quadrant of the image, unless the image is 8i, in which case
whole image will be inverted. */
/* First, get dimensions of image: */
nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp
);
if ( nErrcount == 0 )    /* If that succeeded: */
{
   if ( (nBpp == 8) && (IG_image_is_gray(hIGear) == FALSE) ) /*
If 8i:*/
      {
      /* Invert contrast of whole image: */
      nErrcount = IG_IP_contrast_invert(hIGear,NULL,
IG_CONTRAST_PALETTE);
      }
   else               /* Otherwise: */
      {
      /* Create rectangle of upper right quadrant: */
      rcInvertRect.top    = 0;
      rcInvertRect.left   = nWid / 2;
      rcInvertRect.right  = nWid - 1;
      rcInvertRect.bottom = nHi  / 2;
      /* Invert contrast of upper-right quadrant only: */
```

```
nErrcount = IG_IP_contrast_invert ( hIGear, &rcInvertRect,
                                                IG_CONTRAST_PIXEL );
      }
   }
if ( nErrcount ) { … }    /* If any errors, process them  */
```

IG_IP_contrast_stretch

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_stretch (HIGEAR hIGear,
LPAT_RECT lpRect, AT_MODE nMethodMode );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct defining the rectangle within the image that this function is to operate on. If NULL, the entire image will be operated on. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to alter the pixels themselves (the image bitmap) or the palette

Description:

This function automatically adjusts the contrast of the image so that at least one pixel is completely black and one pixel is completely white. This fills the entire range of the pixel intensities. The original pixel intensities are adjusted linearly between the 2 extremes. (If the image already fills the entire range then the image is not altered.)

Images that use the entire range often appear richer and the colors display more vivid.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR         hIGear;      /* HIGEAR handle of image */
IG_IP_contrast_stretch ( hIGear, NULL,IG_CONTRAST_PIXEL );
```

IG_IP_convert_to_gray

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_convert_to_gray (HIGEAR hIGear );
```

Arguments:

hIGear	HIGEAR handle of image

Description:

This function converts the image referenced by **hIGear** to a grayscale image.

Applicability: All except grayscale

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* HIGEAR handle of image */
IG_IP_convert_to_gray ( hIGear );
```

IG_IP_convolve_matrix

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_convolve_matrix(
```

```
HIGEAR hIGear,
LPAT_RECT lpRect,
LPINT  lpMatrix,
UINT   nMatrixWidth,
UINT   nMatrixHeight,
DOUBLE dblNormalizer,
AT_MODE ColorChannel,
AT_MODE nResultForm,
BOOL   bAddToOrigin
);
```

Arguments:

hIGear	HIGEAR handle of image to be processed
lpRect	Rectangle of image to process; setting to NULL will process the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non - rectangular ROI defined by the mask.
lpMatrix	Far pointer to first INT object in array of kernel elements
nMatrixWidth	A variable of type UINT that holds the width of the array; the maximum value is 16
nMatrixHeight	A variable of type UINT that holds the height of the array; the maximum value is 16
dblNormalizer	A variable of type DOUBLE that contains the value of the normalizer of the kernel
nColorChannel	An AT_MODE constant such as IG_COLOR_COMP_R, _G, _B, or _RGB
nResultForm	An AT_MODE constant, such as IG_CONV_RESULT_8BIT_SIGNED
bAddToOrigin	A variable of type BOOL that tells whether to add the result of the convolution to the pixel values.

Description:

This function convolves the image using a user-defined convolution kernel. The **bAddToOrigin** argument determines whether the result of the convolution is added to the pixel values, or if it replaces the pixel values:

```
if bAddToOrigin = TRUE
   pixel = pixel + result_of_convolution
if bAddToOrigin = FALSE
   pixel = result_of_convolution
```

The result of each convolution is ***multiplied*** by the normalizer, **dblNormalizer**. For kernels that sum to zero, the normalizer is usually set to 1.0. When the sum is not zero, the normalizer's value will depend on the goal of convolution. In a non-weighted averaging convolution the kernel elements are often all ones. In this case the normalizer would be equal to 1/(sum of kernel). Remember that the normalizer is multiplied by the sum of the convolution and not divided into it.

The result of the convolution is an 8-bit image. **nResultForm** determines the type of the format of this new image:

- IG_CONV_RESULT_RAW - the result is stored as is.
- IG_CONV_RESULT_ABS - the absolute value of the signed result is stored.
- IG_CONV_RESULT_8BIT_SIGNED - the result is stored as 8-bit signed values. The MSB is used for the sign.
- IG_CONV_RESULT_SIGN_CENTERED - the result is stored as 8-bit signed, but 0 is equal to 0x7F. Positive numbers are from 0x7E to 0x00. Negative numbers are from 0x80 to 0xFF. This is used for images that are to be used as background tiles or watermarks. The **IG_FX_emboss()** function uses this format.

NOTE: See the Glossary for another explanation of "Convolution." Note that this particular convolution method multiples the results of the convolution by the normalizer, rather than dividing by the convolution, as described in the Glossary.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR       hIGear;     /* HIGEAR handle of image */
UINT         mxConv[5,3];  /* Convolution kernel  */
DOUBLE       dblNormalizer; /* Normalizer of the kernel  */
AT_MODE      nResultForm;    /* Storage format of results */
BOOL        bAddToOrigin;  /* TRUE = add to original pix vals */
AT_RECT      lpRect;        /* rectangle to process  */

mxConv = { 1, 1, 1, 1, 1,
   -2,-2,-2,-2,-2
```

```
    1, 1, 1, 1, 1 };

IG_IP_convolve_matrix ( hIGear, NULL, &mxConv, 5, 3, 1.0,
    IG_COLOR_COMP_RGB, IG_CONV_RESULT_RAW, FALSE );
```

IG_IP_crop

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_crop (HIGEAR hIGear, LPAT_RECT
lpCropRect );
```

Arguments:

hIGear	HIGEAR handle of image
lpCropRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to keep. The remainder of the image is removed and discarded.

Description:

This function crops the image to the image rectangle. Only pixels that fall on or inside the **lpCropRect** rectangle will be kept in the resulting image. The dimensions of the resulting image are then same as that of the **lpCropRect**. The removed parts of the image are discarded.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image */
AT_RECT rcCropRect;    /* Rectangular part remaining  */
AT_DIMENSION nWidth,      /* Original width and height  */
       nHeight;
UINT   nBpp;      /* Bits per pixel (not used) */
AT_DIMENSION nHalfWidth,    /* Will hold half the width */
       nHalfHeight;   /* and half the height    */

/* Set crop rectangle to half the size of the image, centered:
*/
if ( IG_image_is_valid (hIGear) )
   {
   IG_image_dimensions_get ( &hIGear, &nWidth, &nHeight, &nBpp
);
   nHalfWidth  =  nWidth / 2;   nHalfHeight  =  nHeight / 2;
```

```
rcCropRect.left   =  nHalfWidth / 2;
rcCropRect.right  =  rcCropRect.left + nHalfWidth - 1;
rcCropRect.top    =  nHalfHeight / 2;
rcCropRect.bottom =  rcCropRect.top  + nHalfHeight - 1;
   /* Crop the image: */
nErrCount =  IG_IP_crop ( hIGear, &rcCropRect );
}
```

IG_IP_decrypt

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_decrypt ( HIGEAR hIGear, LPAT_RECT
lpRect, AT_MODE nEncryptType, LPSTR lpszPassword );
```

Arguments:

hIGear	HIGEAR handle of image to be decoded
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to decode. Set = NULL for the whole image
nEncrptyType	An IG_ENCRYPT_METHOD_ constant specifying how the image was encoded
lpszPassword	Far pointer to your zero- terminated password string

Description:

This function works on a DIB, not on a file. It decodes an image or a rectangular portion thereof, that was encrypted using **IG_IP_encrypt()**. You must supply both the encryption method and the password that were used in the call to **IG_IP_encrypt()**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;

/* Decrypt the whole hIGear image by method A & with “Top
Secret” password   */
IG_IP_decrypt (hIGear, NULL, IG_ENCRYPT_METHOD_A, “Top
Secret”);
```

IG_IP_deskew_angle_find

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_deskew_angle_find (HIGEAR hIGear,
LPDOUBLE lpAngle);
```

Arguments:

hIGear	HIGEAR handle of image
lpAngle	Far pointer to a DOUBLE variable that will receive the skew angle

Description:

This function determines the skew angle of a 1-bit document image. The angle is returned via the pointer **lpAngle**. You may then use the value of lpAngle with **IG_IP_rotate_any_angle()** function in order to straighten the image.

Applicability: 1bpp indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;  /* HIGEAR handle of image    */
DOUBLE  dblDeskewAngle;  /* Amount of skew in lpRect returned
*/
AT_ERRCOUNT  nErrcount;  /* will tally any IG errors  */
nErrcount = IG_IP_deskew_angle_find ( hIGear, &dblDeskewAngle );
```

IG_IP_deskew_auto

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_deskew_auto (HIGEAR hIGear, DOUBLE
dblAngleThresh, AT_MODE nExpand_clip_option);
```

Arguments:

hIGear	HIGEAR handle of image to be decoded
dblAngleThresh	Do not de-skew if the skew angle is less than this parameter.
nExpand_clip_option	An AT_MODE type variable, either: IG_ROTATE_CLIP or IG_ROTATE_EXPAND

Description:

This function automatically detects the angle of the 1-bit document referenced by **hIGear**, and rotates it so that it is straight. (de-skews it.) However, if the skew angle is less than **dblAngleThresh**, then the image is not de-skewed. If nExpand_clip_option is set to IG_ROTATE_EXPAND, the size of image width or height will be enlarged as necessary to accommodate the document when it has been rotated. Otherwise, any areas of the document that now fall outside the borders of the original width and height of the image, will be cropped.

NOTE: If you want to first detect the angle and then decide whether to rotate it, you can make separate calls to **IG_IP_deskew_angle_find()** and **IG_IP_rotate_any_angle()** functions.

Applicability: 1bpp indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image  */
DOUBLE angle_thresh;/* max angle above which should not perform
skew  */
AT_MODE nExpand_clip_option;/* expand image or clip doc  */
AT_ERRCOUNT nErrcount;    /* will tally returned IG errors */
nErrcount = IG_IP_deskew_auto ( hIGear, 3.0, IG_ROTATE_CLIP);
```

IG_IP_despeckle

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_despeckle ( HIGEAR hIGear, const
LPAT_RECT lpRect );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. If NULL, the operation will be performed on the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.

Description:

Despeckle is used to help reduce the amount of noise in the image. Single pixels and pixel spurs on letters and graphics are removed while leaving the solid areas alone. It is typically used on 1-bit document images.

The despeckle operation performs a 3x3 median filter on the image. For each 3x3 neighborhood of pixels in the original image, a single pixel is produced in the output image. In this case the output is the median of the 9 values in the 3x3 neighborhood.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All except indexed with >1bpp

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;  /* HIGEAR handle of image */
nErrCount = IG_IP_despeckle ( hIGear, NULL );
```

IG_IP_draw_frame

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_draw_frame ( HIGEAR hIGear,
AT_DIMENSION width, AT_MODE nMethod, LPAT_PIXEL lpColor );
```

Arguments:

hIGear	HIGEAR handle of image
width	A variable that holds the width of the frame, in pixels
nMethod	A variable of type AT_MODE that tells ImageGear which IG_DRAW_FRAME_ setting to use. See below.
lpColor	A far pointer to the RGB value that specifies the frame's color

Description:

This function adds a frame (block of solid color on all four sides) to the image referenced by **hIGear**.

If **nMethod** is set to **IG_DRAW_FRAME_EXPAND**, the width and the height of the image will be expanded by 2 times the width of the frame. If **nMethod** is set to **IG_DRAW_FRAME_OVERWRITE**, all four sides of the image will be overwritten by the frame, so that the resulting image has the same width and height as the original image, but the edges of the image are "covered" by the frame.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount;  /* holds tally of IG errors */
AT_DIMENSION width;  /* width, in pixels, of frame  */
AT_MODE  nMethod;  /* expand or overwrite image  */
AT_PIXEL lpColor[256];  /* color of frame; RGB value  */
nErrcount = IG_IP_draw_frame ( hIGear, 5, IG_DRAW_FRAME_EXPAND,
&lpColor[9]);
```

IG_IP_edge_map

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_edge_map ( HIGEAR hIGear, coust
LPAT_RECT lpRect, const AT_MODE nEdgeMapType );
```

Arguments:

hIGear	HIGEAR handle of an image.
lpRect	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangualr ROI defined by the mask.
nEdgeMapType	Type of edge map operation to perform, such as IG_EDGE_OP_LAPLACIAN, IG_EDGE_OP_ROBERTS, etc. The list of available types is in the accucnst.h.

Description:

This function performs the image processing operation specified by the nEdgeMapType argument. The operation is performed upon the rectangular portion of the image specified by lpRect.

An edge map is an image that shows where there are changes in contrast in the original image. Where there are no changes, the resulting image is black. The stronger the contract change, the brighter the resultant image. The different types of edge maps are slight variations of the algorithm used, and produce slightly different results.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image.If the flag is set to TRUE, and a validpointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mak HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask ().

NOTE: See the descriptions of IG_IP_NR_ROI_mask_associate() and IP_IP_NR_ROI_to_HIGEAR_mask () functions for more details.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of an image *?
IG_IP_edge_map ( hIGear, NULL, IG_EDGE_OF_ALPLACIAN ):
```

IG_IP_encrypt

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_encrypt (HIGEAR hIGear, LPAT_RECT
lpRect, AT_MODE nEncryptType, const LPSTR lpszPassword );
```

Arguments:

hIGear	HIGEAR handle of image to be encoded
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to encode. Set = NULL for the whole image.
nEncrptyType	An IG_ENCRYPT_METHOD_ constant specifying the method to be used. See file accucnst.h for IG_ENCRYPT_METHOD_ constants available
lpszPassword	Far pointer to your zero-terminated password string

Description:

This function scrambles an image bitmap, or a rectangular portion thereof. Your password is also stored. To later decode the image using **IG_IP_decrypt()**, you will need to know both the encryption method and the password used in this call.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

NOTE: This function cannot take a non-rectangular ROI for its **AT_RECT** parameter.

Example:

```
HIGEAR  hIGear;
```

```
/* Encrypt the whole hIGear image by method A & with “Top
Secret” password   */
IG_IP_encrypt ( hIGear, NULL, IG_ENCRYPT_METHOD_A, “Top Secret”
);
```

IG_IP_flip

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_flip (HIGEAR hIGear, AT_MODE
nDirection );
```

Arguments:

hIGear	HIGEAR handle of image to flip
nDirection	IG_FLIP_HORIZONTAL or IG_FLIP_VERTICAL, telling whether to flip horizontally or vertically

Description:

Flips the image referenced by **hIGear** either horizontally or vertically. Flipping horizontally exchanges the right-most pixel column of the image bitmap with the left-most. Flipping vertically exchanges the topmost pixel row (raster) of the image bitmap with the bottom-most. The dimensions of the image do not change.

NOTE: If you want to turn the image upside-down (not the same as a vertical flip), use function **IG_IP_rotate_multiple_90()**, with rotation mode set to IG_ROTATE_180.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* Handle of image to flip    */
AT_ERRCOUNT nErrcount;    /* Will hold ret’d count of errors  */
   /* Flip such that top row becomes bottom row, etc.:  */
nErrcount = IG_IP_flip ( hIGear, IG_FLIP_VERTICAL );
```

IG_IP_gaussian_blur

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_gaussian_blur (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const double dblRadius
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to AT_RECT struct specifying a portion of the image to be affected. NULL means entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dblRadius	Defines the neighborhood to be considered for each pixel.

Description:

This function smoothes the images using Gaussian transform. It makes images look softer and slightly out of focus. A specific feature of Gaussian Blur is that it removes the high-frequency component from the image, which is not the case for the **IG_IP_smooth()**.

Parameter dblRadius corresponds to the Standard Deviation (Sigma) in Gaussian transform. It can range from 0.1 to 500. Typical values for high-resolution images range from 1.0 to 2.0. Larger values will cause greater softening. On the other hand, smaller values are faster. The width of the area considered for each pixel is approximately 6 * dblRadius.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;
/* Blur whole image, radius = 2.0 */
nErrCount =IG_IP_gaussian_blur ( hIGear, NULL, 2.0 );
```

IG_IP_histo_clear

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_histo_clear (LPDWORD lpHisto, UINT
nNumberOfBins );
```

Arguments:

lpHisto	Far pointer to an array of bins (each a DWORD) to be cleared
nNumberOfBins	Number of bins to clear

Description:

This function will clear an array of histogram bins. Call this function prior to calling **IG_IP_histo_tabulate()**, unless you mean to accumulate onto existing contents of the bins.

Applicability: N/A (this function does not operate on an image)

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See the example under the **IG_IP_histo_tabulate()** function.

IG_IP_histo_tabulate

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_histo_tabulate (
    HIGEAR hIGear,
    LPDWORD lpHisto,
    UINT   nNumberOfBins,
    LPAT_RECT lpRect,
    UINT   nYIncr,
    const AT_MODE nColorChannel
```

```
);
```

Arguments:

hIGear	HIGEAR handle of image
lpHisto	Far pointer to an array of bins (each a DWORD) in which to tabulate
nNumberOfBins	Number of bins
lpRect	Far pointer to an AT_RECT struct specifying rectangular portion of the image to tabulate for, or NULL for whole image. an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nYIncr	Set = 1 to sample every pixel. Values larger than 1 will skip rasters of the image bitmap
nColorChannel	For 24 bit images selects which channel: IG_COLOR_COMP_R, _G, or _B (IG_COLOR_COMP_RGB not allowed).

Description:

This function produces a histogram of the pixel values occurring in image **hIGear**, or in the rectangular portion specified if **lpRect** is not NULL.

If **nYIncr** = 1, each pixel in the image is examined. The bin corresponding to that pixel value increments. The number of bins must be large enough to hold the entire histogram. If an 8 or 24-bit image, the number of bins must be 256. If a 4-bit or 1-bit image, the number of bins must be 16 or 2 respectively. Note that for a 24-bit image, only a single color channel can be a histogram at one time. Use argument **nColorChannel** to select the channel.

If **nYIncr** is greater than 1, then rasters of the image are skipped. This can be used to speed up the tabulation when the image is large.

You should call **IG_IP_histo_clear()** before calling this function, unless you intentionally mean to accumulate the count onto the existing contents of the bins.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image */
DWORD   dwHistoBins[256];   /* Array of bins for counting */
IG_IP_histo_clear ( &dwHistoBins, 256 ); /* Clear the bins */
/* Tabulate. If a 24-bit image, only the red will be tabulated:
*/
IG_IP_histo_tabulate (hIGear, &dwHistoBins, 256, NULL, 1,
IG_COLOR_COMP_R);
```

IG_IP_maximum

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_maximum (HIGEAR hIGear, const
LPAT_RECT lpRect, AT_DIMENSION nNeighborWidth, AT_DIMENSION
nNeighborHeight);
```

Arguments:

hIGear	HIGEAR handle of image.
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate upon. If NULL, the entire image will be operated upon. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nNeighborWidth	Width of the neighborhood to include in computing each new pixel's value.
nNeighborHeight	Height of the neighborhood to include in computing each new pixel's value.

Description:

This function performs a maximum filter on an image. A maximum filter makes the lighter pixels larger and shrinks the darker ones. The width and height determine the size of each original pixel's neighborhood to use to compute the maximum output pixels. Most applications will find that 3x3 or 5x5 works best.

Only the lpRect of the image is processed. Set lpRect = NULL to process the entire image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All except indexed with >1bpp

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image */
nErrCount = IG_IP_maximum ( hIGear, NULL, 5, 5 );
```

IG_IP_median

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_median (HIGEAR hIGear, const LPAT_RECT
lpRect, AT_DIMENSION nNeighborWidth, AT_DIMENSION
nNeighborHeight );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate upon. If NULL, the entire image will be operated upon. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nNeighborWidth	Width of the neighborhood to include in computing each new pixel's value
nNeighborHeight	Height of the neighborhood

Description:

This function performs a median filter on an image. A median filter is useful for reducing spike or snow-like noise from an image. The width and height determine the size of each original pixel's neighborhood to use to compute the median output pixels. Most applications will find that 3x3 or 5x5 works best.

Only the **lpRect** of the image is processed. Set **lpRect** = **NULL** to process the entire image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned,

ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All except indexed with >1bpp

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image */
nErrCount = IG_IP_median ( hIGear, NULL, 5, 5 );
```

IG_IP_merge

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_merge (
   HIGEAR hImage1,
   HIGEAR hImage2,
   LPAT_RECT lpImageRect2,
   AT_PIXPOS nDstX,
   AT_PIXPOS nDstY,
   AT_MODE nPix_op
   );
```

Arguments:

hIGear1	HIGEAR handle of image
hIGear2	HIGEAR image to be merged into hIGear1; must have same bit depth as hIGear1.
lpImageRect2	Far pointer to an AT_RECT structure specifying the rectangular portion of hIGear2 to merge into hIGear1. upon. Set to NULL if you want to merge the entire image of hIGear2. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non - rectangular ROI defined by the mask.
nDstX	The x coordinate within hIGear1 at which to place the upper - left corner of hIGear2.
nDstY	The y coordinate within hIGear1 at which to place the upper-left corner of hIGear2.
nPix_op	A variable of constant type AT_MODE that specifies what type of arithmetic operation (merge method) to perform on all pixels of hIGear1 that have been intersected with pixels from hIGear2. Examples are IG_ARITH_ADD, which adds the pixel values of both images, and IG_ARITH_SUB, which subtracts the pixel values of hIGear2 from the corresponding pixel values of hIGear1. For the full list of available constants, see accucnst.h or see the description for **IG_clipboard_paste_op_set()** which also uses these constants for full list.

Description:

This function is used to “place” or merge one image into another. The images do not have to be the same size but do have to be the same bit depth. **hImage2** is drawn into **hImage1**. The top left corner of **hImage1** will be placed at **nDstX**, **nDstY** of **hImage1**. Any over-hanging pixels of **hImage2** will be clipped, automatically. The **nPix_op**, which is defined in **accucnst.h**, determines how the pixels are combined.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can merge a rectangular subregion of **HIGEAR2** into HIGEAR1. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions, for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear1,  /* HIGEAR handle of destination image   */
       hIGear2;  /* HIGEAR handle of image to be merged in */
AT_ERRCOUNT nErrcount; /* # of ImageGear errors on stack */
nErrcount = IG_IP_merge ( hIGear1, hIGear2, NULL, 0, 0,
IG_ARITH_OVER );
```

IG_IP_minimum

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_minimum (HIGEAR hIGear, const
LPAT_RECT lpRect, AT_DIMENSION nNeighborWidth, AT_DIMENSION
nNeighborHeight);
```

Arguments:

hIGear	HIGEAR handle of image.
lpRect	Far pointer to an AT_RECT structure specifying the rectangular portion of the image to operate upon. If NULL, the entire image will be operated upon. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nNeighborWidth	Width of the neighborhood to include in computing each new pixel's value.
nNeighborHeigh t	Height of the neighborhood to include in computing each new pixel's value.

Description:

This function performs a minimum filter on an image. A minimum filter makes the lighter pixels smaller and the darker ones larger. The width and height determine the size of each original pixel's neighborhood to use to compute the minimum output pixels. Most applications will find that 3x3 or 5x5 works best.

Only the lpRect of the image is processed. Set lpRect = NULL to process the entire image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the

operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear overrides the settings passed to the AT_RECT structure and uses the non-rectangular ROI defined by the HIGEAR mask. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All except indexed with >1bpp

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image */
nErrCount = IG_IP_minimum ( hIGear, NULL, 5, 5 );
```

IG_IP_NR_ROI_control_get

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_control_get(HIGEAR hIGear,
AT_MODE  nAttributeID, VOID FAR32 *lpData);
```

Arguments:

hIGear	HIGEAR handle of image for which you would like to query ROI settings.
nAttributeID	Set to an AT_MODE constant for the type of attribute you wish to query. See below.
lpData	This returns the current setting of nAttributeID.

Description:

This function will return to you the current setting of any of the non-rectangular ROI control settings. Supply ImageGear with the **HIGEAR** handle of the image you are querying and the attribute (**nAttributeID**) whose setting you would like to query. The ROI settings available at the time of this writing are:

- IG_CONTROL_NR_ROI_DIB: Returns the DIB which is currently set to be used as the mask HIGEAR.
- IG_CONTROL_NR_ROI_REFERENCE_POINT: Returns the reference point of the mask.

- IG_CONTROL_NR_ROI_REFERENCE_POINT_LEFT: Returns left point of the mask.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_TOP: Returns top point of the mask.
- IG_CONTROL_NR_ROI_CONDITION: Queries the "condition" of the mask HIGEAR: whether it is set to be active or not active for the next IP or Clipboard operation.
- IG_CONTROL_NR_ROI_VALIDATE: Returns TRUE if the current ROI mask is valid.

To change these settings and for details on how these controls can be used, see the description for **IG_IP_NR_ROI_control_set()**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
BOOL      bUseNonRect;
AT_POINT  ptReferencePoint;

/* Find out whether the image rectangle for hIGear is set to be
overridden by a non-rectangular ROI    */
nErrcount = IG_IP_NR_ROI_control_get (hIGear,
IG_CONTROL_NR_ROI_STATE, &bUseNonRect);
/* Find out what the reference point for a mask HIGEAR is in the
hIGear */
nErrcount = IG_IP_NR_ROI_control_get (hIGear,
IG_CONTROL_NR_ROI_REFERENCE_POINT, &ptReferencePoint);
```

IG_IP_NR_ROI_control_set

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_control_set( HIGEAR hIGear,
AT_MODE nAttributeID, const LPVOID lpData);
```

Arguments:

hIGear	HIGEAR handle of the image to which you wish make the non-rectangular ROI setting changes.
nAttributeID	Set to an AT_MODE constant for the non-rectangular ROI attribute you would like to set.
lpData	Set to the desired attribute setting.

Description:

This function allows you to set the non-rectangular ROI attributes associated with any HIGEAR that has an associated non-rectangular ROI mask. These attributes are only applicable for non-rectangular ROIs. Use **IG_IP_NR_ROI_to_HIGEAR_mask()** to create the mask image.

All ROI control settings have defined constants in accucnst.h which have a prefix of **IG_CONTROL_NR_ROI_**. The following is a list of each setting available at the time of this writing, and a description of what each does.

- IG_CONTROL_NR_ROI_DIB: Sets the DIB to be used as the mask HIGEAR for the currently loaded HIGEAR image.
- IG_CONTROL_NR_ROI_REFERENCE_POINT: Sets the position within the HIGEAR image at which the upper-left corner of the masking HIGEAR should be placed.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_LEFT: Sets the left point of the mask.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_TOP: Sets the top point of the mask.
- IG_CONTROL_NR_ROI_CONDITION: Sets whether or not ImageGear should override the AT_RECT argument passed to its API. Set to TRUE if you would like ImageGear to use the non-rectangular ROI defined by the mask HIGEAR. Set to FALSE for ImageGear to use the rectangular ROI defined by the current image rectangle.

NOTE: IG_CONTROL_NR_ROI_CONDITION can also be set using **IG_IP_NR_ROI_mask_associate()** function.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
```

```
BOOL      bCondition;
HIGEAR    hIGearMask;]

/* Find out whether the image rectangle for hIGear is set to be
overridden by a non-rectangular ROI   */
nErrcount = IG_IP_NR_ROI_control_set (hIGear,
IG_CONTROL_NR_ROI_DIB, (LPVOID) hIGearMask);
/* Find out what the reference point for a mask HIGEAR is in the
hIGear */
nErrcount = IG_IP_NR_ROI_control_set (hIGear,
IG_CONTROL_NR_ROI_CONDITION, (LPVOID) bCondition);
```

IG_IP_NR_ROI_mask_associate

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_mask_associate(HIGEAR hIGear,
LPAT_NR_ROI_MASK lpMask, BOOL bState);
```

Arguments:

hIGear	HIGEAR handle of the image to associate with a non-rectangular ROI mask.
lpMask	Pass ImageGear a long pointer to a structure of type AT_NR_ROI_MASK that gives the HIGEAR handle of the mask HIGEAR.
bState	Set to TRUE if you want to make the mask "active", meaning that affected IP or clipboard operations will operate on the non-rectangular region only; set to FALSE if you want affected IP or clipboard operations to apply the image rectangle specified by the AT_RECT argument.

Description:

This function will associate a mask **HIGEAR**, as specified by the **AT_NR_ROI_MASK** structure, with the image referenced by **hIGear**. The **bState** argument determines whether or not the ROI defined by **AT_NR_ROI_MASK** should be made active. If you set **bState** to **TRUE**, ImageGear will override the **AT_RECT** argument passed to certain Image Processing and Clipboard API functions in favor of using the non-rectangular ROI. For example, the function **IG_IP_contrast_adjust()** takes an **AT_RECT** as an argument, so that you can adjust the contrast in a rectangular sub-region of an image, or adjust the contrast of the whole image (if you set the **AT_RECT** parameter to **NULL**). If you set **bState** to **TRUE**, when you next call **IG_IP_contrast_adjust()**, its **AT_RECT** argument will be ignored, or “overridden”, and ImageGear will instead use the ROI described by the mask **HIGEAR**.

NOTE: If you provide an invalid mask **HIGEAR**, you will receive the error **IGE_INVALID_MASK_ASSOCIATED**.

No change will take place in the image until you perform an Image Processing or Clipboard operation. When using image processing functions the changes made are permanent if you save the image. For this reason, you may want to keep a copy of the original image so that the user can "undo" an operation.

- The setting for **bState**, which ImageGear stores with the image can also be set using **IG_IP_NR_ROI_control_set()** with the constant **IG_CONTROL_NR_ROI_CONDITION**.
- To reset the reference point in **HIGEAR**, call **IG_IP_NR_ROI_control_set()** with the constant **IG_CONTROL_NR_ROI_REFERENCE_POINT**.
- Call **IG_IP_NR_ROI_mask_unassociate()** to clear the mask **HIGEAR** from its association with **HIGEAR**.

Applicability: All

NOTE: To create a non-rectangular region of interest call **IG_IP_NR_ROI_to_HIGEAR_mask()**function.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
AT_NR_ROI_MASK lpMask;
BOOL      bState;
nErrcount = IG_IP_NR_ROI_mask_associate(hIGear, &lpMask, TRUE);
```

IG_IP_NR_ROI_mask_delete

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_mask_delete (LPAT_NR_ROI_MASK
lpMask);
```

Arguments:

lpMask	A far pointer to the mask structure to delete.

Description:

This function deletes the mask **HIGEAR** created by **IG_IP_NR_ROI_to_HIGEAR_mask()**. When you are done using this mask, and have called the function **IG_IP_NR_ROI_mask_unassociate()**, you should call **IG_IP_NR_ROI_mask_delete()** to delete the mask and free up the memory allocated to it.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

NOTE: See also: **IG_IP_NR_ROI_to_HIGEAR_mask()**, **IG_IP_NR_ROI_mask_associate()**, and **IG_IP_NR_ROI_mask_unassociate()** functions.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
AT_NR_ROI_MASK Mask
nErrcount = IG_IP_NR_ROI_mask_delete(&Mask);
```

IG_IP_NR_ROI_mask_unassociate

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_mask_unassociate(HIGEAR
hIGear);
```

Arguments:

hIGear	HIGEAR handle to the image for which you would like to unassociate a mask HIGEAR.

Description:

This function clears the non-rectangular ROI information from a **HIGEAR** image. This function does not delete the mask **HIGEAR**, it only removes the reference to it from this **HIGEAR**. To delete the mask **HIGEAR**, call **IG_IP_NR_ROI_mask_delete()**.

NOTE: See the description for **IG_IP_NR_ROI_mask_associate()** function for more information.

Applicability: All

NOTE: To delete the mask **HIGEAR**, use **IG_image_delete()** function.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
nErrcount = IG_IP_NR_ROI_mask_unassociate(hIGear);
```

IG_IP_NR_ROI_to_HIGEAR_mask

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_to_HIGEAR_mask(AT_MODE
nSimpleAreaTypeID, LPVOID lpAreaSegmentDesc,LPAT_NR_ROI_MASK
lpNR_ROI);
```

Arguments:

nSimpleAreaTypeID	Set to an AT_MODE constant that describes what kind of non-rectangular region of interest (ROI) you will be passing in. See supported AT_MODE constants below.
lpAreaSegmentDesc	Pass in an array of segment descriptors which will be used to reproduce/ render the non-rectangular ROI as a mask. These segment descriptors can be points in the case of polygons, points and angles in the case of ellipses and so on. The segment descriptors use image coordinates to describe the mask.
lpNR_ROI	ImageGear returns you a structure of type AT_NR_ROI_MASK which contains the HIGEAR handle of the new mask HIGEAR and the "reference point" for its placement within HIGEAR.

Description:

This function is a non-rectangular ROI (region on interest) support function whose purpose is to build a non-rectangular ROI mask from a set of segment descriptors that you pass in, and to return a pointer to a non-rect ROI mask data structure. This non-rect ROI mask data structure (of type AT_NR_ROI_MASK) is then used in conjunction with other API functions to create, associate, modify, and apply the non-rect ROI to the source image.

The **AT_NR_ROI_MASK** structure, shown below, contains two important pieces of information. **ptMaskOffset** describes the coordinates for reference point for the mask **HIGEAR**. **ptMaskOffset** is the (x,y) position in the original **HIGEAR** at which the upper left corner of the mask **HIGEAR** should be placed. The mask **HIGEAR** is actually a rectangle which is calculated by determining the smallest rectangular area that can encompass the entire non-rectangular ROI. We refer to this area as the “bounding rectangle”. Within the mask, which represents the bounding rectangle, a pixel value of 1 indicates that the pixel is within the non-rect ROI; a pixel value of 0 indicates that the pixel is outside the non-rect ROI.

```
typedef struct tag AT_NR_ROI_MASK
{
   AT_POINT ptMaskOffet;
   HIGEAR hMask;

}   AT_NR_ROI_MASK, FAR* LPAT_NR_ROI_MASK;
```

The second member of the mask structure is a **HIGEAR** handle to the actual mask image. The mask HIGEAR is a run length-encoded binary image.

You must also pass this function an argument that specifies whether the region of interest is elliptical, polygonal, or rectangular (the default), using one of the following constants:

- IG_ROI_IS_RECTANGLE
- IG_ROI_IS_ELLIPSE
- IG_ROI_IS_POLYGON

The default ROI type, which is **IG_ROI_IS_RECTANGLE** means that when this mask is associated with and activated for an image, all affected API should use the **AT_RECT** argument that is part of their argument list. If **nSimpleAreaTypeID** is set to **IG_ROI_IS_ELLIPSE** or IG_ROI_IS_POLYGON, all affected API will override their **AT_RECT** arguments and instead look for an associated AT_NR_ROI_MASK.

Applicability: N/A (this function does not operate on an image)

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**. The **lpNR_ROI** parameter will store the mask **HIGEAR** information structure when the function return value is **IGE_SUCCESS**.

NOTE: Use **IG_IP_NR_ROI_mask_associate()** function to associate the mask **HIGEAR** with a **HIGEAR** image.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
AT_NR_ROI_MASK region;
AT_POINT  ROI[];
nErrcount =
   IG_IP_NR_ROI_to_HIGEAR_mask(IG_ROI_IS_POLYGON, &ROI[0],
&region);
```

IG_IP_pseudocolor_limits

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_pseudocolor_limits (HIGEAR hIGear,
LPAT_RGB lpRGB_Low,LPAT_RGB lpRGB_High, AT_PIXEL nLow, AT_PIXEL
nHigh );
```

Arguments:

hIGeaR	HIGEAR handle of 8-bit gray scale image to be colored
lpRGB_Low	Far pointer to an AT_RGB struct (note: B-G-R) specifying the color to apply to all pixels below the nLow value
lpRGB_High	Far pointer to an AT_RGB struct (note: B-G-R) specifying the color to apply to all pixels above the nHigh value
nLow	All pixels below this value are colored
nHigh	All pixels above this value are colored

Description:

This function colors all pixels in an 8-bit gray level image whose values are outside the range **nLow** to **nHigh**. Those above the range receive the color pointed to by **lpRGB_High**, and those below the range receive the color pointed to by **lpRGB_Low**. This function can be used to see how much of the image is saturated or unsaturated.

NOTE: Remember that the order in the RGB structure is B-G-R. See the section Device-Independent Bitmaps (DIB) for more information.

Applicability: 8bpp grayscale

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image  */
AT_RGB  cLowColor, cHighColor; /* The colors to apply */
AT_PIXEL nLow, nHigh;  /* Where to apply them  */

cLowColor.b  = cLowColor.g  = 0;  cLowColor.r  = 255; /* bright
red  */
cHighColor.b = cHighColor.r = 0;  cHighColor.g = 255; /* bright
green*/
   /* Retain image colors for pixel values 25 through 225:
*/
nLow = 25;  nHigh = 225;
IG_IP_pseudocolor_limits ( hIGear, &cLowColor, &cHighColor,
nLow, nHigh );
```

IG_IP_pseudocolor_small_grads

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_pseudocolor_small_grads (HIGEAR
hIGear, UINT nSlope );
```

Arguments:

hIGear	HIGEAR handle of 8-bit gray scale image to be colored
nSlope	An integer from 1 to 255. Higher values increase colors faster.

Description:

This function colors an 8-bit gray level image such that small gradients are exposed. The greater the value of **nSlope**, the faster the color will change for a given rate of change of the pixel value.

Applicability: 8bpp grayscale

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;
if ( IG_image_is_valid(hIGear) )
   if ( IG_image_is_gray(hIGear) )
   IG_IP_pseudocolor_small_grads ( hIGear, 10 );
```

IG_IP_resize

Group: Image Processing

NOTE: Please, use new upgraded function **IG_IP_resize_bkgrnd()**. See note below.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize (HIGEAR hIGear, AT_DIMENSION
nNewWidth, AT_DIMENSION nNewHeight, AT_MODE nInterpMethod );
```

Arguments:

hIGear	HIGEAR handle of image to resize
nNewWidth	Width that the image is to be after resize
nNewHeight	Height that the image is to be after resize
nInterpMethod	IG_INTERPOLATION_NONE, or the interpolation method to use to compute the new pixel values for the resized image. The IG_INTERPOLATION_... constants are listed in file accucnst.h: IG_INTERPOLATION_NONE IG_INTERPOLATION_AVERAGE IG_INTERPOLATION_BILINEAR IG_INTERPOLATION_NEAREST_NEIGHBOR IG_INTERPOLATION_PADDING IG_INTERPOLATION_GRAYSCALE IG_INTERPOLATION_PRESERVE_WHITE IG_INTERPOLATION_PRESERVE_BLACK IG_INTERPOLATION_BICUBIC

Description:

This function will change the dimensions of the image referenced by **hIGear**, by changing the dimensions of the DIB image bitmap. The image data in the bitmap will be stretched or compressed as necessary, to fit the new dimensions. Use **nNewWidth** and **nNewHeight** to specify the new dimensions.

During re-sizing, new pixel values which previously did not exist in the image, may be introduced due to interpolation. If you want to prevent this, such as to preserve the original number of palette entries used, then specify **IG_INTERPOLATION_NONE**. In this case, only pixel values that occur in the original image will result in the resized image.

If you try to apply some interpolation methods to a 4-bit image, you will receive the error **IGE_PROC_INVAL_FOR_BIT_COUNT**. If you try to apply some interpolation methods to an 8-bit indexed image you will receive this same error and also **IGE_PROC_INVAL_FOR_8BIT_INDEXED**.

NOTE: **IG_IP_resize_bkgrnd()** works for 16-bit gray level images using the following resize modes:

IG_INTERPOLATION_NONE
IG_INTERPOLATION_AVERAGE
IG_INTERPOLATION_BILINEAR

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call if there were no errors.

Example:

```
HIGEAR hIGear;    /* Handle of image to be resized  */
AT_DIMENSION nNewWid,    /* New width the image is to be  */
       nNewHi;    /* New height  */
AT_DIMENSION nOldWid,  /* Old width of image  */
       nOldHi,    /* Old height  */
       nBpp;      /* Bits per pixel (not used below) */
AT_ERRCOUNT nErrcount;/* Returned count of errors  */
/* Resize image to 1.5 times its present width, but same height:
*/

/* First, obtain current dimensions:  */
nErrcount = IG_image_dimensions_get ( hIGear1, &nOldWid,
&nOldHi, &nBpp );
if ( nErrcount == 0 )     /* If no errors doing that:  */
{
/* Multiply no. of pixels per row by 1.5:   */
   nNewWid = (nOldWid * 3) / 2;
   nNewHi  =  nOldHi;    /* Keep image the same height  */
   /* Resize, with no interpolation:     */
   nErrcount = IG_IP_resize ( hIGear, nNewWid, nNewHi,
                                          IG_INTERPOLATION_NONE
);
}
if ( nErrcount } { … }   /* If errors anywhere above, process
here  */
```

NOTE: The functionality of this API call has been upgraded and supported by the new function **`IG_IP_resize_bkgrnd()`**. The reason that this new function has been created is that the old function does not allow you to change the background color around the image being resized. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

IG_IP_resize_bkgrnd

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize_bkgrnd (
   HIGEAR hIGear,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod,
   LPAT_PIXEL lpBkgColor
   );
```

Arguments:

hIGear	HIGEAR handle of image to resize
nNewWidth	Width that the image is to be after resize
nNewHeight	Height that the image is to be after resize
nInterpMethod	IG_INTERPOLATION_NONE, or the interpolation method to use to compute the new pixel values for the resized image. The IG_INTERPOLATION_... constants are listed in file accucnst.h. 0 - Standard algorithm 1 - Standard (average) interpolation 2 - Bilinear 3 - Nearest Neighbor interpolation 4 - Resize by adding padding to the image or by cropping the image (this is not an interpolaiton method.) 5 - Scale to Gray 6 - Peserve White 7 - Preserve Black 8 - Bicubic interpolations
lpBkgColor	A far pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image has been resized when use resize with padding method.

Description:

This function will change the dimensions of the image referenced by **hIGear**, by changing the dimensions of the DIB image bitmap. The image data in the bitmap will be stretched or compressed as necessary, to fit the new dimensions. Use **nNewWidth** and **nNewHeight** to specify the new dimensions. Set **lpBkgColor** to the color you would like ImageGear to paint the pixels in the image window which are exposed after the image has been resized. This background color is only applicable when you are resizing the image to a smaller size.

During re-sizing, new pixel values which previously did not exist in the image, may be introduced due to interpolation. If you want to prevent this, such as to preserve the original number of palette entries used, then specify **IG_INTERPOLATION_NONE**. In this case only pixel values that occur in the original image will result in the resized image.

NOTE: IG_IP_resize_bkgrnd() works for 16-bit gray level images using the following resize modes:
IG_INTERPOLATION_NONE
IG_INTERPOLATION_AVERAGE
IG_INTERPOLATION_BILINEAR

Applicability: All

Return Value:

The following table illustrates what happens when you try to apply interpolation (or none) to images of different bit depths.

Bit Depth	None, Padding, Nearest Neighbor	Average and Bilinear Interpolation	Preserve Black, Preserve White, Scale-to-gray	Bicubic Interpolation
1	+	+	+	IG error
4	+	IG error	IG error	IG error
8i	+	IG error	IG error	IG error
8-bit gray level	+	+	IG error	+
9—16-bit gray level	+ (see note below)	+	IG error	+
24 and others	+	+	IG error	+

NOTE: Setting the _PADDING means that if you increase the size of the image, it is padded to the new boundaries. If you display the image, you see white space to the right and bottom of the original image. If you decrease the size of the image with _PADDING, the image is cropped.

Example:

```
HIGEAR hIGear;    /* Handle of image to be resized  */
AT_DIMENSION nNewWid,   /* New width the image is to be   */
       nNewHi;    /* New height   */
AT_DIMENSION nOldWid,   /* Old width of image  */
       nOldHi,    /* Old height    */
       nBpp;      /* Bits per pixel (not used below) */
AT_ERRCOUNT nErrcount; /* Returned count of errors  */
AT_RGB rgb;       /* color of background; RGB value  */
/* Enlarge image to 1.5 times its present width, but same
height, padding with the green color up to right: */
/* First, obtain current dimensions:     */
nErrcount = IG_image_dimensions_get ( hIGear1, &nOldWid,
&nOldHi, &nBpp );
if ( nErrcount == 0 )    /* If no errors doing that:  */
{
/* Multiply no. of pixels per row by 1.5:  */
```

```
    nNewWid = (nOldWid * 3) / 2;
    nNewHi  =  nOldHi;     /* Keep image the same height  */
    /* Resize, with padding:     */
    rgb.r = 0;
    rgb.g = 255;
    rgb.b = 0;
    nErrcount = IG_IP_resize_bkgrnd ( hIGear, nNewWid,
    nNewHi,IG_INTERPOLATION_PADDING, (LPAT_PIXEL)&rgb);
}
if ( nErrcount } { … }     /* If errors anywhere above, process
here  */
```

IG_IP_resize_bkgrnd_ex

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize_bkgrnd_ex(
[IN/OUT]HIGEAR hIGear,
[IN]AT_DIMENSION nNewWidth,
[IN]AT_DIMENSION nNewHeight,
[IN]AT_MODE InterpMethid,
[IN]AT_MODE lpBkgColor,
[IN]DWORD dwFlags,
[IN]INT   nValue
);
```

Arguments:

hIGear	HIGEAR handle of the image to resize
newWidth	The width of the resized image
nNewHeight	The height of the resized image
nInterpMethod	IG_INTERPOLATION_NONE, or the interpolation method to use to compute the new pixel values for the resized image. The IG_INTERPOLATION_... constants are listed in accucnst.h file. 0 - Standard algorithm 1 - Standard (average) interpolation 2 - Bilinear 3 - Nearest Neighbor interpolation 4 - Resize by adding padding to the image or by cropping the image (this is not an interpolaiton method.) 5 - Scale to Gray 6 - Peserve White 7 - Preserve Black 8 - Bicubic interpolations
lpBkgColor	A far pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image is resized using padding method.
dwFlags	The contents of this parameter depends upon the value of nInterpMethod. This is currently not used.
nValue	The contents of this parameter depends upon the value of nInterpMethod. This is currently used only if nInterpMethod is one of the following: ■ IG_INTERPOLATION_GRAYSCALE - nValue can be from 0 to 100. It takes the proportion of pixels from entry 1 to entry 0 (white/black) ■ IG_INTERPOLATION_PRESERVE_WHITE - nValue can be from 0 to 100. It indicates the threshold value of the amount of white color to include. ■ IG_INTERPOLATION_PRESERVE_BLACK - nValue can be from 0 to 100. It indicates the threshold value of the amount of black color to include.

Description:

This function changes the dimensions of the image referenced by hIGear by changing the dimensions of the DIB image bitmap. The image data in the bitmap is stretched or compressed as necessary, to fit the new dimensions. Use nNewWidth and nNewHeight to specify the new dimensions. Set lpBkgColor to the color you would like ImageGear to paint the pixels in the image window that are exposed after the image has been resized. This background color is only applicable when you are resizing the image to a smaller size.

During re-sizing, new pixel values that previously did not exist in the image, may be introduced due to interpolation. If you want to prevent this, (to preserve the original number of palette entries used, for example) specify `IG_INTERPOLATION_NONE`. In this case, only the pixel values that occur in the original image will result in the resized image.

NOTE: `IG_IP_resize_bkgrnd_ex()` works for 16-bit gray level images using the following resize modes:

`IG_INTERPOLATION_NONE`
`IG_INTERPOLATION_AVERAGE`
`IG_INTERPOLATION_BILINEAR`

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Interpolation Specs:

Bit Depth	None, Padding, Nearest Neighbor	Average and Bi-linear Interpolation	Preserve Black, Preserve White, Scale-to-gray	Bicubic Interpolation
1	+	+	+	IG error
4	+	IG error	IG error	IG error
8i	+	IG error	IG error	IG error
8-bit gray level	+	+	IG error	+
9-16-bit gray level	+(see note below)	+	IG error	+
24 and others	+	+	IG error	+

NOTE: Setting the `_PADDING` means that if you increase the size of the image, it is padded to the new boundaries. If you display the image, you see white space to the right and bottom of the original image. If you decrease the size of the image with `_PADDING`, the image is cropped.

Example:

```
HIGEAR hIGear;      /* Handle of image to be resized */
AT_DIMENSION nNewWid,     /* New width the image is to be */
       nNewHi;      /* New height  */
AT_DIMENSION nOldWid,     /* Old width of image  */
       nOldHi,      /* Old height  */
       nBpp;        /* Bits per pixel (not used below)  */
AT_ERRCOUNT nErrcount;   /* Returned count of errors  */
AT_RGB rgb;  /* color of background; RGB value */
/* Enlarge image to 1.5 times its present width, but same
height, padding with the green color up to right:  */
/* First, obtain current dimensions:    */
nErrcount = IG_image_dimensions_get ( hIGear1, &nOldWid,
&nOldHi, &nBpp );
if ( nErrcount == 0 )     /* If no errors doing that: */
{
/* Multiply no. of pixels per row by 1.5:    */
   nNewWid = (nOldWid * 3) / 2;
   nNewHi  =  nOldHi;      /* Keep image the same height  */
   /* Resize, with padding:       */
   rgb.r = 0;
   rgb.g = 255;
   rgb.b = 0;
   nErrcount = IG_IP_resize_bkgrnd_ex ( hIGear, nNewWid,
   nNewHi, IG_INTERPOLATION_PADDING, (LPAT_PIXEL)&rgb, 0, 0 );
}
if ( nErrcount } { … }     /* If errors anywhere above, process
here  */
```

IG_IP_resize_ex

Group: Image Processing

NOTE: Please, use new upgraded function **`IG_IP_resize_bkgrnd_ex()`**. See note below.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize_ex(
[IN/OUT]HIGEAR hIGear,
[IN]AT_DIMENSION nNewWidth,
[IN]AT_DIMENSION nNewHeight,
[IN]AT_MODE nInterpMethod,
[IN]DWORD dwFlags,
[IN]INT nValue
);
```

Arguments:

hIGear	HIGEAR handle of the image to resize
nNewWidth	Width to which the image is to be resized.
nNewHeight	Height to which the image is to be resized.
nInterpMethod	IG_INTERPOLATION_NONE, or the interpolation method to use to compute the new pixel values for the resized image. The IG_INTERPOLATION_... constants are listed in file accucnst.h. 0 - Standard algorithm 1 - Standard (average) interpolation 2 - Bilinear 3 - Nearest Neighbor interpolation 4 - Resize by adding padding to the image or by cropping the image (this is not an interpolaiton method.) 5 - Scale to Gray 6 - Peserve White 7 - Preserve Black 8 - Bicubic interpolations
dwFlags	The contents of this parameter depends upon the value of nInterpMethod. It is currently not used.
nValue	The contents of this parameter depends upon the value of nInterpMethod. It is currently used only if nInterpMethod is used. ■ IG_INTERPOLATION_GRAYSCALE - nValue can be from 0 to 100. It takes the proportion of pixels from entry 1 to entry 0 (white/black) ■ IG_INTERPOLATION_PRESERVE_WHITE - nValue can be from 0 to 100. It indicates the threshold value of the amount of white color to include. ■ IG_INTERPOLATION_PRESERVE_BLACK - nValue can be from 0 to 100. It indicates the threshold value of the amount of black color to include.

Description:

This function changes the dimensions of the image referenced by hIGear by changing the dimensions of the DIB image bitmap. The image data in the bitmap will be stretched or compressed as necessary, to fit the new dimensions. Use nNewWidth and nNewHeight to specify the new dimensions.

During re-sizing, new pixel values that previously did not exist in the image may be introduced due to interpolation. If you want to prevent this, such as to preserve the original number of palette entries used, then specify IG_INTERPOLATION_NONE. In this case, the only pixel values that occur in the original image will result in the resized image.

If you try to add interpolation to a 4-bit image, you will receive the error IGE_PROC_INVAL_FOR_BIT_COUNT. If you try to use interpolation for an 8-bit indexed image you will receive this same error and also IGE_PROC_INVAL_FOR_8BIT_INDEXED.

NOTE: **IG_IP_resize_bkgrnd_ex()** works for 16-bit gray level images using the following resize modes:
IG_INTERPOLATION_NONE
IG_INTERPOLATION_AVERAGE
IG_INTERPOLATION_BILINEAR

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;     /* Handle of image to be resized */
AT_DIMENSION nNewWid,    /* New width the image is to be  */
       nNewHi;     /* New height   */
AT_DIMENSION nOldWid,   /* Old width of image  */
       nOldHi,     /* Old height   */
       nBpp;       /* Bits per pixel (not used below)  */
AT_ERRCOUNT nErrcount; /* Returned count of errors  */
/* Resize image to 1.5 times its present width, but same height:
*/

/* First, obtain current dimensions:     */
nErrcount = IG_image_dimensions_get ( hIGear1, &nOldWid,
&nOldHi, &nBpp );
if ( nErrcount == 0 )     /* If no errors doing that:  */
{
/* Multiply no. of pixels per row by 1.5:    */
   nNewWid = (nOldWid * 3) / 2;
   nNewHi  =  nOldHi;       /* Keep image the same height */
   /* Resize, with no interpolation:      */
   nErrcount = IG_IP_resize_ex ( hIGear, nNewWid,
nNewHi,IG_INTERPOLATION_NONE, 0, 0 );
}
if ( nErrcount } { … }     /* If errors anywhere above, process
here  */
```

NOTE: The functionality of this API call has been upgraded and supported by the new function **IG_IP_resize_bkgrnd_ex()**. The reason that this new function has been created is that the old function does not allow you to change the background color around the image being resized. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

IG_IP_RGB_to_hue

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_RGB_to_hue (const LPAT_RGB lpRGB,
LPDOUBLE lpHue );
```

Arguments:

lpRGB	A long pointer to a structure of type AT_RGB containing three bytes of color information.
lpHue	A long pointer to a double containing the hue value that corresponds to the color specified by `lpRGB`

Description:

This function will convert a 24-bit RGB value to a hue value. You pass it a pointer to the **`AT_RGB`** structure of your choice, and it returns the hue angle (0.0 to 360) to you. This is useful before calling **`IG_FX_chroma_key()`** which requires a hue angle as one of its arguments. (ImageGear's pixel access functions only read and write RGB values.) You can use them to read the RGB value of a "background" pixel in the image which you wish to make transparent.

Applicability: N/A (this function does not operate on an image)

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

See the example for **`IG_FX_chroma_key()`** function.

IG_IP_rotate_any_angle

Group: Image Processing

NOTE: Please, use new upgraded function **`IG_IP_rotate_any_angle_bkgrnd()`**. See note below.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_any_angle ( HIGEAR hIGear,
DOUBLE angle, AT_MODE rotate_mode );
```

Arguments:

hIGear	HIGEAR handle of image
angle	Angle to rotate, in degrees; this may be set to a positive or negative value.
rotate_mode	An integer variable of type AT_MODE defined in accucnst.h. The value can be either: ■ IG_ROTATE_CLIP - clip the rotated image to keep the image bitmap the same size or ■ IG_ROTATE_EXPAND - expand the size of the bitmap if necessary to retain the entire rotated image.

Description:

This function rotates the image about its center through the angle indicated by **angle**. (**angle** must be expressed in degrees, such as 45.0 to rotate 45 degrees.) **rotate_mode** is examined to determined whether to clip the image or resize the bitmap.

If you have called this function with the **IG_ROTATE_EXPAND** setting, you can use **IG_IP_rotate_compute_size()** to calculate the new dimensions of the bitmap.

Applicability: All

NOTE: The functionality of this API call has been upgraded and supported by the new function **IG_IP_rotate_any_angle_bkgrnd()**. The reason that this new function has been created is that the old function does not allow you to change the background color around the image being rotated. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* HIGEAR handle of image to rotate  */
DOUBLE  angle;    /* Angle to rotate counterclockwise  */
AT_ERRCOUNT nErrcount;   /* Returned number of errors       */
 …
angle  = 45.0           /* Set angle to rotate  */
nErrcount = IG_IP_rotate_any_angle ( hIGear, angle,
IG_ROTATE_CLIP );
```

IG_IP_rotate_any_angle_bkgrnd

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_any_angle_bkgrnd ( HIGEAR
hIGear, DOUBLE angle, AT_MODE rotate_mode, LPAT_PIXEL
lpBkgrndColor );
```

Arguments:

hIGear	HIGEAR handle of image
angle	Set to a positive or negative value for angle of rotation.
rotate_mode	An integer variable of type AT_MODE defined in accucnst.h. The value can be either: ■ **IG_ROTATE_CLIP -** clip the rotated image to keep the image bitmap the same size **or** ■ **IG_ROTATE_EXPAND -** expand the size of the bitmap if necessary to retain the entire rotated image.
lpBkgrndColor	A far pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image has been rotated.

Description:

This function rotates the image about its center through the angle indicated by **angle**. (**angle** must be expressed in degrees, such as 45.0 to rotate 45 degrees.) **rotate_mode** is examined to determined whether to clip the image or resize the bitmap. Set **lpBkgrndColor** to the color that you would like to display in the background that was previously covered by the image before it was rotated.

If you have called this function with the **IG_ROTATE_EXPAND** setting, you can use **IG_IP_rotate_compute_size()** to calculate the new dimensions of the bitmap.

NOTE: You can only rotate PDF/PS images in 90 degree increments.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image to rotate  */
DOUBLE angle;    /* Angle to rotate counterclockwise  */
AT_ERRCOUNT nErrcount;   /* Returned number of errors  */
AT_RGB rgb;         /* color of background; RGB value */
```

```
 …
rgb.r = 0;
rgb.g = 255;
rgb.b = 0;
angle  = 45.0     /* Set angle to rotate  */
nErrcount = IG_IP_rotate_any_angle_bkgrnd ( hIGear, angle,
IG_ROTATE_CLIP, (LPAT_PIXEL)&rgb);
```

IG_IP_rotate_any_angle_ex

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_any_angle_ex ( HIGEAR
hIGear, DOUBLE angle, AT_MODE rotate_mode, LPAT_PIXEL
lpBkgrndColor, AT_MODE interpolation );
```

Arguments:

hIGear	HIGEAR handle of image
angle	Set to a positive or negative value for angle of rotation.
rotate_mode	An integer variable of type AT_MODE defined in accucnst.h. The value can be either: ■ **IG_ROTATE_CLIP -** clip the rotated image to keep the image bitmap the same size **or** ■ **IG_ROTATE_EXPAND -** expand the size of the bitmap if necessary to retain the entire rotated image.
lpBkgrndColor	A far pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image has been rotated.
interpolation	Interpolation to use for rotation (IG_INTERPOLATION_NONE, IG_INTERPOLATION_BILINEAR, or IG_INTERPOLATION_BICUBIC)

Description:

This function rotates the image about its center through the angle indicated by **angle**. (**angle** must be expressed in degrees, such as 45.0 to rotate 45 degrees.) **rotate_mode** is examined to determined whether to clip the image or resize the bitmap. Set **lpBkgrndColor** to the color that you would like to display in the background that was previously covered by the image before it was rotated. The interpolation parameter controls what type of interpolation, if any, is used for rotation.

If you have called this function with the **IG_ROTATE_EXPAND** setting, you can use **IG_IP_rotate_compute_size()** to calculate the new dimensions of the bitmap.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image to rotate  */
DOUBLE angle;    /* Angle to rotate counterclockwise  */
AT_ERRCOUNT nErrcount;   /* Returned number of errors  */
AT_RGB rgb;          /* color of background; RGB value */
 …
rgb.r = 0;
rgb.g = 255;
rgb.b = 0;
angle  = 45.0     /* Set angle to rotate  */
nErrcount = IG_IP_rotate_any_angle_bkgrnd ( hIGear, angle,
IG_ROTATE_CLIP, (LPAT_PIXEL)&rgb);
IG_INTERPOLATION_BILINEAR
```

IG_IP_rotate_compute_size

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_compute_size ( HIGEAR hIGear,
DOUBLE dblAngle, AT_MODE nRotateMode, LPAT_DIMENSION lpWidth,
LPAT_DIMENSION lpHeight );
```

Arguments:

hIGear	HIGEAR handle of image to compute size of
dblAngle	The angle (in degrees) that you want to rotate the image
nRotateMode	An integer variable of type AT_MODE defined in accucnst.h. The value can be either: ■ **IG_ROTATE_CLIP -** clip the rotated image to keep the image bitmap the same size **or** ■ **IG_ROTATE_EXPAND -** expand the size of the bitmap if necessary to retain the entire rotated image.
lpWidth	A long pointer in which is returned the new width of the bitmap after rotation
lpHeight	A long pointer in which is returned the new height of the bitmap after rotation.

Description:

This function can be used to compute the new height and width of **hIGear** after it has been rotated using **IG_IP_rotate_any_angle()** or **IG_IP_rotate_any_angle_bkgrnd()** functions. You may use it before rotation, to determine the dimensions of the new rotated image. This way, you can estimate the amount of memory that will be needed to hold the new image.

This function is actually only relevant if the mode of rotation is set to **IG_ROTATE_EXPAND**. Otherwise, the height and width would remain the same after rotation. (with clipping.)

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image  */
DOUBLE dblAngle;             /* Angle of rotation  */
AT_DIMENSION Width, Height;  /* New height and width of image
*/
AT_ERRCOUNT nErrcount;      /* Totally of ImageGear errors    */
AT_DIMENSION szAngleWidth[46];   /* Hold widths at diff angles
*/
INT  i;
/* loop will store the resulting widths of hIGear when it is set
to angles between 0 and 45 degrees. Results stored to array
*/
for (i = 0; i < 46; i++)
{
nErrcount = IG_IP_rotate_compute_size (hIGear, (DOUBLE)i,
IG_ROTATE_EXPAND, &Width, &Height);
szAngleWidth[i] = Width;
```

```
}
```

IG_IP_rotate_multiple_90

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_multiple_90 (HIGEAR hIGear,
AT_MODE nMult_90_Mode );
```

Arguments:

hIGear	HIGEAR handle of image to rotate
nMult_90_Mode	One of IG_ROTATE_0, IG_ROTATE_90, IG_ROTATE_180, or IG_ROTATE_270

Description:

This function will rotate the image referenced by **hIGear** clockwise about its center. The rotation must be a multiple of 90 degrees. If the rotation is either 90 or 270 degrees, the previous width of the image becomes the new height, and the previous height of the image becomes the new width. **IG_ROTATE_0** does nothing and is included for completeness only.

To rotate by an arbitrary angle that may not be a multiple of 90 degrees, use functions **IG_IP_rotate_any_angle()**or **IG_IP_rotate_any_angle_bkgrnd()**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
/* HIGEAR     handle of image to rotate   */
AT_MODE       nRotateMode; /* Will be set to an IG_ROTATE_…
constant */
BOOL      bClockwiseFlag;/* Flag indicating direction to rotate
*/
AT_ERRCOUNTnErrcount;    /* Returned number of errors */
 …
/* If flag is TRUE, rotate 90 clockwise, else 90
counterclockwise:  */
if ( bClockwiseFlag )
   { nRotateMode = IG_ROTATE_270; } /* Go clockwise 90 degrees
*/
else
   { nRotateMode = IG_ROTATE_90;  } /* clockwise 90   */
nErrcount = IG_IP_rotate_multiple_90 ( hIGear, nRotateMode );
```

IG_IP_sharpen

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_sharpen (HIGEAR hIGear, const
LPAT_RECT lpRect, const INT nSharpFactor );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying a portion of the image to be affected. NULL means entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nSharpFactor	Factor indicating the degree for increasing image sharpness. Valid range is 1 to 5.

Description:

This function causes the dark side of a contrast edge to become darker and the light side lighter. Areas without contrast are not altered. **nSharpFactor** should be in the range 1 to 5. The higher the value, the more sharpening will be applied.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

NOTE: See also function **IG_IP_smooth()**.

Applicability: All except indexed

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;     /* Handle of image to sharpen  */
nErrCount = IG_IP_sharpen ( hIGear, NULL, 2 );/* Sharpen whole
image, factor = 2*/
```

IG_IP_smooth

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_smooth( HIGEAR hIGear, const LPAT_RECT
lpRect, const INT nSmoothFactor );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to an AT_RECT struct specifying the portion of image to be affected. NULL means whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nSmoothFactor	Factor indicating degree of decrease to sharpness or increase to smoothness wanted. Valid range is 1 to 4.

Description:

This function makes images look softer and slightly out of focus. This can be used to reduce the graininess of an image. **nSmoothFactor** can range from 1 to 4. Larger values will cause greater softening.

The pixel neighborhood considered by this function is 3x3, 5x5, 7x7, or 9x9 for **nSmoothFactor** = 1, 2, 3, or 4 respectively. Therefore a smaller value is faster. **nSmoothFactor** = 1 or 2 is recommended.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to **TRUE**, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

NOTE: See the function **IG_IP_sharpen()**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;    /* Handle of image to smooth   */
nErrCount=IG_IP_smooth ( hIGear, NULL, 2 ); /* Smooth whole
image, factor = 2*/
```

IG_IP_swap_red_blue

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_swap_red_blue ( HIGEAR hIGear );
```

Arguments:

hIGear	HIGEAR handle of image

Description:

This function reverses the color sequence in the pixels of image **hIGear's** image bitmap. If the sequence is Blue-Green-Red (the standard sequence for a DIB), it is reversed to Red-Green-Blue. If Red-Green-Blue, each pixel is reversed to Blue-Green-Red. This function is typically used on 24-bit RGB images, but it can operate on other image types as well.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* HIGEAR handle of 24-bit image */
IG_IP_swap_red_blue ( hIGear );
```

IG_IP_thumbnail_create

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_thumbnail_create (
    HIGEAR hOrignalImage,
    LPHIGEAR lphNewThumbnail,
    AT_DIMENSION nNewWidth,
    AT_DIMENSION nNewHeight,
    AT_MODE nInterpMethod
    );
```

Arguments:

hOriginalImage	HIGEAR handle of image to create thumbnail image of
lphNewThumbnail	Far pointer to a variable of type HIGEAR to receive the HIGEAR handle of the created thumbnail image
nNewWidth	Specifies width wanted for the new image
nNewHeight	Specifies height wanted for the new image
nInterpMethod	Specifies interpolation method to use for computing values of the pixels in the new image

Description:

This function creates a new image and provides you its new HIGEAR handle. The new image is a resize of the original image. This new image will be **nNewWidth** by **nNewHeight**.

The interpolation parameter is used to determine how the new pixels are to be computed from the original one. **nInterpMethod** = **IG_INTERPOLATION_NONE** specifies that no interpolation is to be used. In this case, only pixel values that occur in the original image will occur in the new image. See file accucnst.h to learn of any other interpolation methods available.

NOTE: Please see description of **IG_IP_resize()** and other **IG_IP_resize_...** functions for interpolation modes description as well as for information on how this function works with different bits per pixels.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear,    /* HIGEAR handle of image to create thumbnail
of */
hIGearThumb;      /* Will be handle of the 64 x 64 thumbnail
image */
ErrCount = IG_IP_thumbnail_create ( hIGear, &hIGearThumb, 64,
64,IG_INTERPOLATION_NONE );
```

IG_IP_thumbnail_create_ex

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_thumbnail_create_ex
[IN]HIGEAR hOriginalImage,
[OUT]LPHIGEAR lphNewThumbnail,
[IN]AT_DIMENSION nNewWidth,
[IN]AT_DIMENSION nNewHeight,
[IN]AT_MODE nInterpMethod,
[IN]DWORD dwFlags,
[IN]INT   nValue
);
```

Arguments:

hOriginalImage	HIGEAR handle of the image for which to create the thumbnail image
lphNewThumbnail	Far pointer to a variable of type HIGEAR to receive the HIGEAR handle of the created thumbnail image
nNewWidth	Specifies the `width` specified for the new image
nNewHeight	Specifies the `height` specified for the new image
nInterpMethod	Specifies interpolation method to use for computing values of the pixels in the new image
dwFlags	The contents of this parameter depends upon the value of `nInterpMethod`. This is currently not used.
nValue	The contents of this parameter depends upon the value of `nInterpMethod`. This is used only if `nInterpMethod` is: ■ `IG_INTERPOLATION_GRAYSCALE` - `nValue` can be from 0 to 100. It takes the proportion of pixels from entry 1 to entry 0 (white/black) ■ `IG_INTERPOLATION_PRESERVE_WHITE` - `nValue` can be from 0 to 100. It indicates the threshold value of the amount of white color to include. ■ `IG_INTERPOLATION_PRESERVE_BLACK` - `nValue` can be from 0 to 100. It indicates the threshold value of the amount of black color to include

Description:

This function creates a new image and provides its new `HIGEAR` handle. The new image is a resize of the original image. This new image will be `nNewWidth` by `nNewHeight`.

The interpolation parameter is used to determine how the new pixels are to be computed from the original one. `nInterpMethod = IG_INTERPOLATION_NONE` specifies that no interpolation is to be used. In this case, only pixel values that occur in the original image will occur in the new image. See the `accucnst.h` file for other interpolation methods that are currently available.

NOTE: See the description of **`IG_IP_resize()`** and other **`IG_IP_resize_...`** functions for interpolation modes as well as for information on how this function works with different bits per pixels.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear,    /* HIGEAR handle of image to create thumbnail
of */
hIGearThumb;      /* Will be handle of the 64 x 64 thumbnail
image */
IG_IP_thumbnail_create_ex( hIGear, &hIGearThumb, 64, 64,
IG_INTERPOLATION_NONE, 0, 0 );
;
```

IG_IP_transform_with_LUT

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_transform_with_LUT (HIGEAR hIGear,
LPAT_RECT lpRect, LPAT_PIXEL lpLUTr, LPAT_PIXEL lpLUTg,
LPAT_PIXEL lpLUTb, AT_MODE nColorMode );
```

Arguments:

hIGear	HIGEAR handle of image to transform
lpRect	Rectangular portion of the image to process; set to NULL for entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
lpLUTr	Far pointer to a user-supplied Look Up Table for transforming the red component, or for transforming the pixel value if the image is less than 24-bit.
lpLUTg	Far pointer to a user-supplied Look Up Table for transforming the green component, or for transforming the pixel value if the image is 24-bit.
lpLUTb	Far pointer to a user-supplied Look Up Table for transforming the blue component, or for transforming the pixel value if the image is 24-bit.
nColorMode	A variable of type AT_MODE (IG_COLOR_COMP_) that tells which color channel to use, or to use all three.

Description:

This function transforms the pixel values of the image referenced by **hIGear**, using a LUT. The pixels from hIGear are used as indices into the LUT. The entry in the LUT at this position is placed into the new image. For 24-bit images, the three channels each have their own LUT. You can point all three LUT parameters to the same LUT. This will process all three channels the same. For 8-bit gray level images, only the lpLUTr LUT parameter is used and the pixel value replaced from the LUT is as follows and the other two are ignored.

```
new pixel value  = RedLUT[ old pixel value ].
```

This function, like other ImageGear Image Processing and Clipboard API calls, takes an **AT_RECT** structure as an argument, so that you can process a rectangular subregion of an image. (*See above.*) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to **TRUE**, indicating that a mask **HIGEAR** should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the **AT_RECT** structure and use the non-rectangular ROI defined by the mask **HIGEAR**. To create a non-rectangular region of interest, call **IG_IP_NR_ROI_to_HIGEAR_mask()**.

NOTE: Please see the descriptions of **IG_IP_NR_ROI_mask_associate()** and **IG_IP_NR_ROI_to_HIGEAR_mask()** functions for more details.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;         /* HIGEAR handle of image    */
AT_PIXEL LUTred[256];  /* Look up pixels, or red if 24-bit, here
*/
AT_PIXEL LUTgreen[256];/* Look up green if 24-bit, here   */
AT_PIXEL LUTblue[256]; /* Look up blue if 24-bit, here    */

IG_IP_transform_with_LUT ( hIGear,
(LPAT_PIXEL)&LUTred,(LPAT_PIXEL)&LUTgreen,
(LPAT_PIXEL)&LUTblue, IG_COLOR_COMP_RGB );
```

IG_IP_unsharp_mask

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_unsharp_mask (HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const double dblRadius,
   const UINT nAmount,
   const UINT nThreshold,
   const AT_MODE nColorChannel );
```

Arguments:

hIGear	HIGEAR handle of image
lpRect	Far pointer to AT_RECT struct specifying a portion of the image to be affected. NULL means entire image. Before ImageGear performs this operation, it will check to see if internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dblRadius	Defines the neighborhood to be considered for each pixel.
nAmount	Amount of sharpening, in percents.
nThreshold	Minimal difference between a pixel and its neighbors at which the pixel will be modified.
nColorChannel	Color channel to which the transform shall be applied.

Description:

Unsharp masking filter is used for image sharpening and edge enhancement. The algorithm works by substracting a smoothed version of the image (“unsharp”) from the original image.

Parameter `dblRadius` can range from 0.1 to 500. Typical values for high-resolution images range from 1.0 to 2.0. Use larger values of `dblRadius` for thicker edges. The smaller `dblRadius` value is, the faster is this processing .

Parameter `nAmount` can range from 1 to 500. Typical values for photographic images range from 100 to 200. Use bigger values for greater edge contrast.

Parameter `nThreshold` can range from 1 to 500. Typical values for photographic images are between 0 and 20. Use `nThreshold` to apply sharpening only to those areas where contrast changes significantly. Flat areas will remain unchanged. A proper selection of `nThreshold` allows to ehnance edges on the image while leaving insufficient details, such as noise or grain of the photographic film, unchanged.

Parameter `nColorChannel` specifies the channel to which the transform shall be applied. The default value is `IG_COLOR_COMP_ALL`: apply transform to all color channels of the image. To run **UnsharpMask** process on the Intensity channel of the image, convert the image to YUV colorspace, run **UnsharpMask** filter with `nColorChannel` = `IG_COLOR_COMP_YUV_Y`, and convert back to original colorspace.

This function, like other ImageGear Image Processing and Clipboard API calls, takes `AT_RECT` structure as an argument, so that you can process a rectangular subregion of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to `TRUE`, indicating that a mask `HIGEAR` should be used with the image. If the flag is set to `TRUE`, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the `AT_RECT` structure and use the non-rectangular ROI defined by the mask `HIGEAR`. To create a non-rectangular region of interest, call **`IG_IP_NR_ROI_to_HIGEAR_mask()`**.

Applicability: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;
/* Sharpen whole image, radius = 2.0, amount = 150, threshold =
10 */
nErrCount =IG_IP_unsharp_mask ( hIGear, NULL, 2.0, 150, 10,
IG_COLOR_CHANNEL_ALL );
```

IG_lic_solution_name_set

Group: Utility

Declaration:

```
AT_ERRCODE ACCUAPI IG_lic_solution_name_set (LPCHAR
lpSolutionName)
```

Arguments:

lpSolutionName	The license solution name.

Description:

This function provides **solution name** for your ImageGear license. Please see the section ImageGear Licensing API.

Bits Per Pixel: N/A

Return Value:

Returns the result code of this function call.

IG_load_alpha_mode_get

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_alpha_mode_get (enumIGAlphaMode
*lpAlphaMode );
```

Arguments:

lpAlphaMode	Far pointer to an enumIGAlphaMode variable to receive the current Alpha chanel loading mode setting.

Description:

This function retrieves the last setting made by calling function **IG_load_alpha_mode_set()**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
enumIGAlphaMode AlphaMode;
IG_load_alpha_mode_get ( &AlphaMode );
```

Sample: Filter

IG_load_alpha_mode_set

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_alpha_mode_set (enumIGAlphaMode
AlphaMode );
```

Arguments:

AlphaMode	IG_ALPHA_MODE_KEEP (default) to load Alpha channel if it is present; IG_ALPHA_MODE_IGNORE to ignore Alpha channel.

Description:

Instructs ImageGear to load or ignore alpha channel when loading an image that contains one.

NOTE: See also **IG_load_alpha_mode_get()** function.

Bits Per Pixel: all

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
/* Ignore Alpha channel when loading: */
IG_load_alpha_mode_set (IG_ALPHA_MODE_IGNORE );
```

Sample: Filter

IG_load_auto_detect_get

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_auto_detect_get ( AT_MODE
nFormatType, LPBOOL lpOnOff);
```

Arguments:

nFormatType	An AT_MODE constant such as IG_FORMAT_BMP telling which filter to obtain the detect-state of
lpOnOff	Far pointer to a variable of type BOOL in which will be returned the current state of the filter: enabled (TRUE), or disabled (FALSE)

Description:

Obtains the current state of the format filter indicated by **nFormatType**.

NOTE: See also function **IG_load_auto_detect_set()**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
BOOL bEnabled; /* Will be TRUE if this file format can be
accessed */
```

```
nErrCount = IG_load_auto_detect_get ( IG_FORMAT_TIF , &bEnabled
);
```

IG_load_auto_detect_set

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_auto_detect_set(AT_MODE
nFormatType, BOOL bOnOff);
```

Arguments:

nFormatType	An AT_MODE constant such as IG_FORMAT_BMP telling which filter to set on (enabled) or off (disabled)
bOnOff	Set the filter enabled (TRUE), or disabled (FALSE). The default value is TRUE.

Description:

Enable or disable a format filter. If disabled, that file format type cannot be accessed by the IG_load_…() functions.

NOTE: See also function **IG_load_auto_detect_get()**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

NOTE: The ASCII filter is turned off by default. To turn it on set:
`IG_load_auto_detect_set(IG_FORMAT_TXT, TRUE);`

Example:

```
BOOL bEnabled; /* If FALSE this file format will be disabled */
nErrCount = IG_load_auto_detect_set ( IG_FORMAT_TIF , FALSE );
```

IG_load_CCITT_FD

Group: Image Loading

NOTE: Please, use new upgraded function **IG_load_raw_FD()**. See note below.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_CCITT_FD ( AT_INT fd, AT_DIMENSION
nWidth, AT_DIMENSION nHeight, AT_MODE nType, AT_MODE nFillOrder,
LPHIGEAR lphIGear);
```

Arguments:

fd	File Descriptor handle of the open file containing raw Fax data to be loaded
nWidth	Number of pixels in each row of data to be read
nHeight	Number of rows of data
nType	Specifies the type of compression: IG_COMPRESSION_CCITT_G3, IG_COMPRESSION_CCITT_G4 or IG_COMPRESSION_G32D. There is no default for this propery.
nFillOrder	IG_FILL_MSB or IG_FILL_LSB, specifying whether the most-significant-bit-first or least-significant-bit-first. There is no default for this propery.
lphIGear	Returned ImageGear handle of resulting HIGEAR image

Description:

This function creates an ImageGear image from raw CCITT Compressed data located in the file whose File Descriptor handle is **fd**. The pointer must be positioned at the start of the data. (For example, your application should read or seek past any header that is present.) You must specify the type, G3 or G4, by means of argument **nType**, and you must specify the fill order, most-significant-bit-first or least-significant-bit-first, by argument **nFillOrder**. The most common fill order is most-significant-bit-first or IG_FILL_MSB.

The width and height of the image are specified by **nWidth** and **nHeight**. The handle of the resulting new ImageGear image is returned in the **HIGEAR** variable pointed to by **lphIGear**. The resulting image is always 1-bit.

This function is used when you have a non-standard or proprietary G3 or G4 compressed image file and you know the details of the header. There are literally hundreds of different types of image files that fall into this category. In order to be able to successfully read an image of this type you must know enough about the header to find where the height and width are stored. You can usually look at the header with a hex dump utility and see where these values are stored. Once you are able to read past the header plus get the dimensions of the image, you can then use this function. You can experiment with the other settings until the image is read correctly.

NOTE: The functionality of this API call has been upgraded and supported by the new function **IG_load_raw_FD()**. The reason that this new function has been created to expand the number of raw image types you can load into ImageGear. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

NOTE: G3 compressed images are always 1728 pixels wide. Since G3 files usually have a special code at the end of the image that ImageGear will detect, you can set the height to a value greater than the expected height of the image and it will be corrected once the end of image marker is detected. For G4 files the height and width must be known and in all cases the file pointer must be at the start of the compressed image when this function is called.

Bits Per Pixel: 1

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;         /* Will receive HIGEAR image handle   */
INT  fd;             /* File Descriptor handle  */
AT_DIMENSION nWid,      /* Pixels per row of data to read */
     nHi;            /* Rows to read  */
AT_ERRCOUNT nErrcount;   /* Returned count of errors */
nErrcount = IG_load_CCITT_FD ( fd, nWid, nHi,
IG_COMPRESSION_CCITT_G3, IG_FILL_MSB, &hIGear );
```

IG_load_CCITT_mem

Group: Image Loading

NOTE: Please, use new upgraded function **IG_load_raw_mem()**. See note below.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_CCITT_mem (
   LPVOID lpImage,
   AT_UINT nSize,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   AT_MODE nType,
```

```
AT_MODE nFillOrder,
LPHIGEAR lphIGear
);
```

Arguments:

lpImage	Far pointer to memory area from which to load
nSize	Size of image in memory.
nWidth	Number of pixels in each row of data to be read
nHeight	Number of rows of data
nType	Specifies the type of compression: `IG_COMPRESSION_CCITT_G3`, `IG_COMPRESSION_CCITT_G4`, or `IG_COMPRESSION_CCITT_G32D`. There is no default for this propery.
nFillOrder	`IG_FILL_MSB` or `IG_FILL_LSB`, specifying whether most-significant bit, first or least-significant bit first. There is no default for this propery.
lphIGear	Returned ImageGear handle of resulting `HIGEAR` image

NOTE: The functionality of this API call has been upgraded and supported by the new function **`IG_load_raw_mem()`**. The reason that this new function has been created is that the old function restricted you to loading raw images that are stored with **`CCITT`** formatting.
In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

Description:

This function operates similarly to function **`IG_load_CCITT_FD()`**. See the description under that function. Note that **`lpImage`** must point to the start of the actual data (not to the start of any header information that may be present).

Bits Per Pixel: 1

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;          /* Will receive HIGEAR image handle  */
LPVOID lpPtrToImage;     /* Pointer to image in memory   */
AT_UINT nMemImageSize;    /* Size of image in memory */
AT_DIMENSION nWid,        /* Pixels per row of data to read  */
    nHi;               /* Rows to read   */
AT_ERRCOUNT   nErrcount;   /* Returned count of errors  */
```

```
nErrcount = IG_load_CCITT_mem ( lpPtrToImage, nMemImageSize,
nWid, nHi, IG_COMPRESSION_CCITT_G3, IG_FILL_MSB, &hIGear );
```

IG_load_color_reduction_get

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_color_reduction_get (LPAT_MODE
lpColorReduceMode );
```

Arguments:

lpColorReduceMode	Far pointer to an AT_MODE variable to receive the current IG_LOAD_COLOR setting

Description:

This function retrieves the last setting made by calling function **IG_load_color_reduction_set()**. If no color reduction is in effect, **IG_LOAD_COLOR_DEFAULT** is returned.

Bits Per Pixel: 4, 8, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_MODE nLoadReduceMode;   /* Will receive the IG_LOAD_COLOR_
constant */
IG_load_color_reduction_get ( &nLoadReduceMode );
```

IG_load_color_reduction_set

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_color_reduction_set ( AT_MODE
nColorReduceMode );
```

Arguments:

nColorReduceMode	One of `IG_LOAD_COLOR_8`, `IG_LOAD_COLOR_4`, or `IG_LOAD_COLOR_1`, else `IG_LOAD_COLOR_DEFAULT` if no color reduction is wanted.

Description:

Instructs ImageGear to perform color reduction to reduce the number of Bits Per Pixel whenever loading an image whose bit depth is greater than that specified by **`nColorReduceMode`**. The bit depth is reduced to 8, 4, or 1 as specified. Call with **`nColorReduceMode`** = **`IG_LOAD_COLOR_DEFAULT`** to disable color reduction.

NOTE: See also **`IG_load_color_reduction_get()`** function.

Bits Per Pixel: 4, 8, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
/* Reduce 24-bit images to 8-bit when loading: */
IG_load_color_reduction_set ( IG_LOAD_COLOR_8 );
```

IG_load_extra_mode_get

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_extra_mode_get (enumIGExtraMode
*lpExtraMode );
```

Arguments:

lpExtraMode	Far pointer to an enumIGExtraMode variable to receive the current Extra chanel loading mode setting.

Description:

This function retrieves the last setting made by calling function **`IG_load_extra_mode_set()`**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
enumIGExtraMode ExtraMode;
IG_load_extra_mode_get ( &ExtraMode );
```

Sample: Filter

IG_load_extra_mode_set

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_extra_mode_set (enumIGExtraMode
ExtraMode );
```

Arguments:

ExtraMode	IG_ALPHA_MODE_KEEP (default) to load Extra channel if it is present; IG_ALPHA_MODE_IGNORE to ignore Extra channel.

Description:

Instructs ImageGear to load or ignore extra channels when loading an image that contains any.

NOTE: See also **IG_load_extra_mode_get()** function.

Bits Per Pixel: all

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
/* Ignore Extra channel when loading: */
IG_load_extra_mode_set (IG_ALPHA_MODE_IGNORE );
```

IG_load_FD

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_FD (
    AT_INT fd,
    LONG lOffset,
    UINT nPage,
    UINT nTileNum,
    LPHIGEAR lphIGear
    );
```

Arguments:

fd	File Descriptor Handle of file containing the image to be loaded.
lOffset	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. `lOffset` is usually 0.
nPageNum	Page number to load if this is a multi- page (multi- image) file. Note that page numbers begin at 1, not 0. Set `nPageNum` to 1 if this is not a multi-page file.
nTileNum	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image
lphIGear	Page Far pointer to `HIGEAR` object to which this function will return the `HIGEAR` handle of the image loaded

Description:

This function loads an image from a file into memory and creates a DIB for the image. Unlike **`IG_load_file()`**, this function is used when the file is already open and you have its File Descriptor handle (**`fd`**). The **`HIGEAR`** handle, which ImageGear assigns for the loaded image, is returned to you via argument **`lphIGear`**. The file indicated by **`fd`** may be in any format recognized by ImageGear. **`IG_load_FD()`** will determine the format by inspecting the file's header section. See File Format Reference.

NOTE: Simply loading the file does not cause it to be displayed. Refer to **`IG_dspl_image_draw()`** and related routines, for how to display an image once it is in memory. See also **`IG_load_file_display()`**.

`lOffset` represents the number of bytes, positive or negative, from the position in the file currently pointed to by `fd`. The `fd` may have been moved around a few times so that it is no longer pointing to the beginning of the file. Be sure to keep this in mind as you set the value of `lOffset`.

The **`nPageNum`** argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set **`nPageNum`** to 1 for a non-multi-page file.

NOTE: If you set `nPageNum` to < 1, ImageGear will default the value to 1; if you set `nPageNum` to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the `nTileNum` parameter as well.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

NOTE: If you wish to make a subsequent call to **`IG_info_get_FD_ex()`**, you must first move the file pointer (of loaded image) to the beginning of the file, or you will receive an error. This happens because after an image is loaded, the file pointer is positioned at the end of the image in the file. To avoid the error:

- Call **`IG_load_FD()`**.
- Call the appropriate C function, that will set the pointer back to the beginning of the image's header information.
- Call **`IG_info_get_FD_ex()`**.

Example:

```
HIGEAR hIGear;          /* handle ret'd by IG_load_FD   */
INT  fd;              /* DOS File Descriptor   */
LONG lOffset;          /* offset to image in file   */
UINT nPageNum;         /* will be 0 for this call  */
AT_ERRCOUNT nErrcount;    /* to test for errors  */
fd = _lopen ( "picture.bmp", OF_READ );   /* open file  */
nPagenum  = 1;                          /* not a multi-page file
*/
lOffset    = 0;          /* access file from start  */
/* Load image, and obtain its HIGEAR handle:    */
nErrcount = IG_load_FD ( fd, lOffset, nPageNum, 1, &hIGear );
```

IG_load_FD_CB

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_FD_CB (
    AT_INT fd
    LONG      lOffset,
    UINT      nPageNum,
    UINT      nTileNum,
    LPFNIG_RASTER_SET lpfnRasterSet,
    LPFNIG_DIB_CREATE lpfnDIBCreate,
    LPVOID    lpPrivateData
    );
```

Arguments:

fd	File Descriptor handle
lOffset	Offset within file at which the image file begins (normally 0)
nPageNum	Page number to load. Set to 1 or greater for a multi-page file; 1 for a non-multi page file.
nTileNum	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image.
lpfnRasterSet	Far pointer to callback func to be called after each raster line is read
lpfnDIBCreate	Far pointer to callback func to be called after the file header has been read
lpPrivateData	Far pointer to a private data area. This pointer will be passed to the callback functions.

Description:

This function loads from an already open file whose File Descriptor handle is **fd**. However, it is the responsibility of your two callback functions, **lpfnDIBCreate** and **lpfnRasterSet**, to create the DIB or other structure you want. Your **lpfnDIBCreate** callback function is called after the file's header has been read. Then your **lpfnRasterSet** callback function is called for each raster line read. See the descriptions under function types **LPFNIG_DIB_CREATE** and **LPFNIG_RASTER_SET** for how these callback functions are called.

The **nPageNum** argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set **nPageNum** to 1 for a non-multi-page file. (Remember that page numbers begin at **1**, not **0**.)

If you want a **HIGEAR** handle for the DIB your callback functions have created, you can obtain one (after the load is complete) by calling function **IG_image_DIB_import()**.

NOTE: Set full CMYK support to get DIB and rasters in CMYK color format.

Bits Per Pixel: 1, 4, 8, 9-6g, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

IG_load_FD_CB_ex

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_FD_CB_ex(
AT_INT fd,
LONG lOffset,
UINT nPage,
UINT nTileNum,
LPFNIG_RASTER_SET lpfnRasterSet,
LPFNIG_DIB_CREATE_EX lpfnDIBCreateEx,
LPVOID lpPrivateData
   );
```

Arguments:

fd	File descriptor
lOffset	Offset within file at which the image file begins (normally 0)
nPage	Page number to load. Set to 1 or greater for a multi-page file; 1 for a non-multi-page file.
nTileNum	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image.
lpfnRasterSet	Pointer to callback function to be called after each raster line is read
lpfnDIBCreateEx	Pointer to callback function to be called after the file header has been read
lpPrivateData	Pointer to a private data area. This pointer will be passed to the callback functions.

Description:

This function loads from an already open file whose File Descriptor handle is `fd`. However, it is the responsibility of your two callback functions, `lpfnDIBCreate` and `lpfnRasterSet`, to create the DIB or other structure you want. Your `lpfnDIBCreate` callback function is called after the file's header has been read.

Then your lpfnRasterSet callback function is called for each raster line read. See the descriptions under function types LPFNIG_DIB_CREATE_EX and **LPFNIG_RASTER_SET** for how these callback functions are called.

The nPageNum argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set nPageNum to 1 for a non-multi-page file. (Remember that page numbers begin at 1, not 0.)

If you want a HIGEAR handle for the DIB your callback functions have created, you can obtain one (after the load is complete) by calling function **IG_image_DIB_import()**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HFILE fd;               /* File descriptor */
DWORD dwPrivate[10];    /* Some private data */
fd = _lopen("test.jpg", OF_READ);
if (fd != HFILE_ERROR)
{
    nErrcount = IG_load_FD_CB_ex(
        fd,               /* File descriptor             */
        0L,               /* Offset to image             */
        1,                /* Page number to load         */
        1,                /* Reserved. Always set to 1   */
        MyRasterSet,      /* Called for each raster line */
        MyDIBCreateEx,    /* Called after header is read */
        dwPrivate);       /* Callback data               */
    _lclose(fd);
}

AT_ERRCOUNT ACCUAPI MyDIBCreateEx(
    LPVOID             lpPrivate, /* Private data passed in  */
    const HIGDIBINFO   hDIB       /* DIB info object for DIB */
    )
{
    /* Get info about image and allocate storage here */
    return 0;
}

AT_ERRCOUNT ACCUAPI MyRasterSet(
    LPVOID           lpPrivate,   /* Private data passed in  */
    const LPAT_PIXEL lpRaster,    /* Raster line to set      */
    AT_PIXPOS        row,         /* Y position in the image */
    DWORD            rasterSize   /* Size of the raster line */
    )
{
```

```
    /* Do something with incoming raster data here */
    return 0;
}
```

IG_load_file

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_file ( const LPSTR szFilename,
LPHIGEAR lphIGear );
```

Arguments:

szFilename	Far pointer to filename (you may include path with filename) of image file to load into memory
lphIGear	Far pointer to HIGEAR object in which to return the ImageGear handle of the image

Description:

This function loads an image file into memory and creates a DIB for the image. The handle which ImageGear assigns for this image is returned to you in argument **lphIGear**. The file named by filename may be in any format recognized by ImageGear. IG_load_file() will determine the format by inspecting the file's header section. See File Format Reference Chapter.

NOTE: Note that simply loading the file does not cause it to be displayed. Refer to **IG_dspl_image_draw()** and related routines, for how to display an image once it is in memory. See also **IG_load_file_display()**.

NOTE: Some file formats, such as TXT, JPEG, and others, may be loaded with additional control, using **IG_fltr_ctrl_get()** and **IG_fltr_ctrl_set()**. See the description of these functions also in Using Format Filters API for Filter Control section.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

NOTE: This function does not support loading multiple images from a multi-page file. If used on a multi-page file, it will only load the first image (page). To load multiple images, use **IG_load_FD()** or **IG_load_mem()**.

Example:

```
HIGEAR hIGear            /* Will hold handle returned by
IG_load_file*/
AT_ERRCOUNT nErrcount;    /* Count of errs on stack upon ret
from func*/
/* Load image file “picture.bmp” from current directory,
creating DIB */
/* and obtaining the image’s ImageGear handle: */
nErrcount = IG_load_file ( “picture.bmp” , &hIGear );
/* this lines demonstrates loading an Internet file  */
nErrCount = IG_load_file( “http://www.accusoft.com/images/
image1.bmp”, &hIGear )
```

IG_load_file_display

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_file_display ( const LPSTR
lpszFileName, DWORD dwGrpID, HWND hWnd, HDC hDC,
LPFNIG_LOAD_DISP lpfnLoadDisp,LPVOID lpPrivate, LPHIGEAR
lphIGear );
```

Arguments:

lpszFileName	Far pointer to filename (including path if desired) of file to load and display
dwGrpID	Display group identifier that should be used for display operations.
hWnd	Handle of window where to draw image.
hDC	Device Context of device or window to display image in
lpfnLoadDisp	Far pointer to a callback function (or name of callback function) to call when image has been loaded, but before it is displayed
lpPrivate	Far pointer to a private data area. This pointer will be passed to the callback function when it is called
lphIGear	Far pointer to a variable of type `HIGEAR` to hold the returned ImageGear `HIGEAR` handle of the newly loaded image

Description:

This function is called to load and then simultaneously display an image. Since you may want to set display attributes prior to displaying, ImageGear first loads the image's header, creating its **`HIGEAR`** handle, then calls your callback function (with the image's **`HIGEAR`** handle and your `lpPrivate` pointer) so you can set display attributes, device rectangle, image rectangle, or perform other operations. When your callback function returns, ImageGear then displays the image, one raster line at a time, as the image is loaded.

See the description for callback type **`LPFNIG_LOAD_DISP`**, and see also the section Displaying Images for a discussion of display attributes and how to set them.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See the example code in the section Working with ImageGear Callback Functions.

IG_load_mem

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_mem (
   LPVOID lpImage,
   AT_UINT nSize,
   UINT nPageNum,
```

```
UINT nTileNum,
LPHIGEAR lphIGear
);
```

Arguments:

lpImage	Far pointer to memory location of an image file that is currently in memory
nSize	Size of image in memory.
nPageNum	If a multi-page file, this is page number to load. Otherwise, it is set to 1.
nTileNum	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image.
lphIGear	Far pointer to `HIGEAR` object in which to return ImageGear handle of the newly loaded image

Description:

This function is very similar in operation to **`IG_load_file()`**, except that the file to load from is in memory instead of on a mass storage device. That is, the entire image file (even if a multi-page file) including header, is in memory. The format must be one of the file formats recognized by ImageGear. See File Format Reference.

Argument `lpImage` is a **`FAR`** pointer to the start of the image file in memory. **`dwSize`** is the size of the entire file (even if a multi-page file). For a multi-page file, **`nPageNum`** is the page number to load. Note that page numbers in multi-page files begin at **1**, not **0**. Set **`nPageNum`** = 1 if the file is a non-multi-page file.

This function creates a DIB for the image and loads the image into it. The handle which ImageGear assigns for this image is returned to you via argument **`lphIGear`**.

NOTE: Note that simply loading the file does not cause it to be displayed. Refer to function **`IG_dspl_image_draw()`** for how to display an image once it is in memory.

NOTE: If you set `nPageNum` to < 1, ImageGear will default the value to 1; if you set `nPageNum` to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the `nTileNum` parameter as well.

Bits Per Pixel:

1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* handle ret'd by ImageGear  */
char far * lpWhereFile;   /* ptr to image file in mem   */
AT_UINT nWholeSize;   /* Size of image in memory */
UINT    nPageNum;   /* will be 0 for this call    */
AT_ERRCOUNT nErrcount;  /* to test for errors         */
nPageNum     = 1;       /* not a multi-page file      */
lpWhereFile  = ...;      /* where mem image of file begins */
dwWholeSize  = ...;     /* size of whole mem image    */
nErrcount = IG_load_mem ( lpWhereFile, nWholeSize, nPageNum, 1,
&hIGear );
```

IG_load_mem_CB

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_mem_CB (  LPVOID lpImage, AT_UINT
nSize, UINT nPageNum, UINT nTileNum, LPFNIG_RASTER_SET
lpfnRasterSet, LPFNIG_DIB_CREATE lpfnDIBCreate, LPVOID
lpPrivate );
```

Arguments:

lpImage	Far pointer to file image in memory
nSize	Size of image in memory.
nPageNum	If a multi-page file, this is page number to load. Otherwise, it is set to 1.
nTileNum	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image.
lpfnRasterSet	Far pointer to callback func to be called after each raster line is read
lpfnDIBCreate	Far pointer to callback func to be called after the file header has been read
lpPrivate	Far pointer to a private data area. This pointer will be passed to the callback functions

Description:

The operation of this function is similar to that of **IG_load_FD_CB()**, except that the file image is in memory beginning at location ***lpImage**, rather than in a file. See the description under function IG_load_FD_CB(). See also function **IG_load_mem()**.

NOTE: If you set **nPageNum** to < 1, ImageGear will default the value to 1; if you set **nPageNum** to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the **nTileNum** parameter as well.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
LPFNIG_DIB_CREATE MyDIB_Create;     /* Create-DIB callback */
LPFNIG_RASTER_SET MyRasterSet;    /* Raster-Set callback    */
LPVOID cPointerToImage;    /* Ptr to start of file image in
memory */
AT_UINT nMemImageSize; /* Size of image in memory */
static char cPrivate[200];  /* Can hold data to pass to
callbacks  */
AT_ERRCOUNT nErrcount;  /* Returned error count  */
nErrcount = IG_load_mem_CB ( cPointerToImage, nMemImageSize, 1,
1,MyRasterSet, MyDIB_Create, (LPVOID) cPrivate );
if ( nErrcount )
{ … }  /* Either callback might have forced an error on stack
*/
/* See also the examples under LPFNIG_DIB_CREATE and
LPFNIG_RASTER_SET. */
```

IG_load_mem_CB_ex

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_mem_CB_ex(
LPVOID lpImage,
AT_UINT nImageSize,
UINT nPage,
UINT nReserved,
LPFNIG_RASTER_SET lpfnRasterSet,
LPFNIG_DIB_CREATE_EX lpfnDIBCreateEx,
LPVOID lpPrivateData
   );
```

Arguments:

lpImage	Pointer to file image in memory
nImageSize	Size of image in memory
nPage	If a multi-page file, this is page number to load. Otherwise, it should be set to 1.
nReserved	Reserved, should be set to 0 for now.
lpfnRasterSet	Pointer to callback function to be called after each raster line is read
lpfnDIBCreateEx	Pointer to callback function to be called after the file header has been read
lpPrivateData	Pointer to a private data area. This pointer will be passed to the callback functions.

Description:

The operation of this function is similar to that of **`IG_load_FD_CB_ex()`**, except that the file image is in memory beginning at location `lpImage`, rather than in a file. See the description under function **`IG_load_FD_CB_ex()`**. See also function IG_load_mem().

NOTE: If you set `nPageNum` to < 1, ImageGear will default the value to 1; if you set `nPageNum` to greater than the number of pages in the document, ImageGear will default the value to the last page number.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

See example for **`IG_load_FD_CB_ex()`**.

IG_load_raw_FD

Group: Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_raw_FD(
   AT_INT fd,
```

```
AT_DIMENSION nWidth,
AT_DIMENSION nHeight,
UINT   nBitsPerPixel,
AT_MODE nFillOrder,
AT_ MODE nCompression,
LPHIGEAR lphIGear
);
```

Arguments:

fd	File Descriptor handle to an image that you would like to load
nWidth	The width, in pixels, of the image
nHeight	The height, in pixels, of the image
nBitsPerPixel	The Bits Per Pixel of the raw data to load
nFillOrder	Set to the fill order used in the image: Least Significant Bit first (LSB) or Most Significant Bit first (MSB). Use one of the ImageGear - defined constants:`IG_FILL_LSB` or `IG_FILL_MSB`. ■ For G3/G4 compressed data, this parameter specifies the Least/Most significant bit. ■ For 16 bit grayscale uncompressed images, this parameter specifies the Least/Most significant byte. ■ For any other bit depths and compressions, this parameter is ignored.
nCompression	Set to an ImageGear- defined compression constant. The names of such constants begin with the prefix `IG_COMPRESSION_` and are defined in `accucnst.h`.
lphIGear	Returns you a `HIGEAR` handle to the raw image data just loaded.

Description:

This function creates an ImageGear image from the raw image data of the file whose File Descriptor handle is **`fd`**. A raw image file contains no header or identifying information. You must supply this function with all of the information needed to correctly parse the image data, including the compression, byte fill order, width, height, and bit depth. As of this writing, this function can be used to read raw image data from the following types of files: CCITT Group 3 & 4, and raw uncompressed data.

The pointer must be positioned at the start of the data. (For example, your application should read or seek past any header that is present.)

Tell ImageGear the width, height, bit depth, compression, and fill order of the image using **`nWidth`**, **`nHeight`**, **`nBitsPerPixel`**, **`nFillOrder`**, and **`nCompression`**, respectively. The handle of the resulting new ImageGear image is returned in the **`HIGEAR`** variable pointed to by **`lphIGear`**.

Additionally, you can specify row alignment of the image using `ALIGNMENT` image control parameter. See RAW format reference in File Format Reference Chapter for more information.

In all cases the file pointer must be at the start of the compressed image when this function is called.

NOTE: For uncompressed images only, ImageGear's Load Raw functions consider the coordinates (0,0) to refer to the lower-left corner of the bitmap.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear;      /* Will receive HIGEAR image handle */
INT    fd;         /* File Descriptor handle    */
AT_DIMENSION nWid,      /* Pixels per row of data to read, */
nHi;                      /* Rows to read       */
AT_ERRCOUNT nErrcount;      /* Returned count of errors */

nErrcount = IG_load_raw_FD ( fd, nWid, nHi, 1, IG_FILL_MSB,
IG_COMPRESSION_CCITT_G3, &hIGear );
```

IG_load_raw_file

Group: Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_raw_file(
   const LPSTRlpszFileName,
   LONG   lOffset,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   UINT   nBitsPerPixel,
   AT_MODE nFillOrder,
   AT_MODE nCompression,
   LPHIGEAR lphIGear
   );
```

Arguments:

lpszFileName	Name of the file
lOffset	Offset in the file where the raw image data starts from
nWidth	The width, in pixels, of the image
nHeight	The height, in pixels, of the image
nBitsPerPixel	The Bits Per Pixel of the raw data to load
nFillOrder	Set to the fill order used in the image: Least Significant Bit first (LSB) or Most Significant Bit first (MSB). Use one of the ImageGear-defined constants: `IG_FILL_LSB` or `IG_FILL_MSB`. ■ For G3/G4 compressed data, this parameter specifies the Least/Most significant **bit**. ■ For 16-bit grayscale uncompressed images, this parameter specifies the Least/Most significant **byte**. ■ For any other bit depths and compressions, this parameter is ignored.
nCompression	Set to an ImageGear-defined compression constant. The names of such constants begin with the prefix `IG_COMPRESSION_` and are defined in `accucnst.h`.
lphIGear	Returns you a `HIGEAR` handle to the raw image data just loaded.

Description:

This function loads a raw (no header) image data file from disk. It performs the same operation as the new API functions **`IG_load_raw_mem()`** and **`IG_load_raw_FD()`**. Please see the descriptions for these API for more details.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

IG_load_raw_mem

Group: Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_raw_mem( LPVOID lpImage, AT_UINT
nImageSize, AT_DIMENSION nWidth, AT_DIMENSION nHeight, UINT
nBitsPerPixel, AT_MODE nFillOrder, AT_MODE nCompression,
LPHIGEAR lphIGear );
```

Arguments:

lpImage	Far pointer to memory area from which to load
nImageSize	Size of image in memory.
nWidth	The width, in pixels, of the image.
nHeight	The height, in pixels, of the image
nBitsPerPixel	The Bits Per Pixel of the raw data to load
nFillOrder	Set to the fill order used in the image: Least Significant Bit first (LSB) or Most Significant Bit first (MSB). Use one of the ImageGear-defined constants:`IG_FILL_LSB` or `IG_FILL_MSB`. ■ For G3/G4 compressed data, this parameter specifies the Least/Most significant **bit**. ■ For 16-bit grayscale uncompressed images, this parameter specifies the Least/Most significant **byte**. ■ For any other bit depths and compressions, this parameter is ignored.
nCompression	Set to an ImageGear-defined compression constant. The names of such constants begin with the prefix `IG_COMPRESSION_` and are defined in `accucnst.h`.
lphIGear	Returns you a `HIGEAR` handle to the raw image data just loaded.

Description:

This function operates similarly to function **`IG_load_raw_FD()`**. See the description under that function.

NOTE: Note that **`lpImage`** must point to the start of the actual data (not to the start of any header information that may be present).

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
HIGEAR hIGear;     /* Will receive HIGEAR image handle  */
LPVOID lpPtrToImage;  /* Pointer to image in memory  */
AT_UINTnMemImageSize;/* Size of image in memory */
AT_DIMENSION nWid,     /* Pixels per row of data to read  */
       nHi;      /* Rows to read */
AT_ERRCOUNT nErrcount;    /* Returned count of errors  */
```

```
nErrcount = IG_load_raw_mem ( lpPtrToImage, nMemImageSize, nWid,
nHi, 16, IG_FILL_LSB, IG_COMPRESSION_NONE, &hIGear );
```

IG_load_rect_get

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_rect_get ( LPAT_PIXPOS lpX,
LPAT_PIXPOS lpY, LPAT_DIMENSION lpWidth, LPAT_DIMENSION
lpHeight );
```

Arguments:

lpX	Far pointer to variable of type AT_PIXPOS to receive X coordinate of load rectangle
lpX	Far pointer to AT_PIXPOS variable to receive Y coordinate of load rectangle
lpWidth	Far pointer to variable of type AT_DIMENSION to receive width of load rectangle
lpHeight	Far pointer to variable of type AT_DIMENSION to receive height of load rectangle

Description:

This function obtains the current load rectangle, as set in the last call to **IG_load_rect_set()**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_PIXPOS nXLoadCrop,    /* X coordinate of load crop rectangle
*/
       nYLoadCrop;     /* Y coordinate  */
AT_DIMENSION nWidCrop,     /* Width of load crop rectangle  */
       nHiCrop;     /* Height   */
IG_load_rect_get ( &nXLoadCrop, &nYLoadCrop, &nWidCrop,
&nHiCrop );
```

IG_load_rect_set

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_rect_set ( AT_PIXPOS nX, AT_PIXPOS
nY, AT_DIMENSION nWidth, AT_DIMENSION nHeight );
```

Arguments:

nX	X coordinate of load rectangle
nY	Y coordinate of load rectangle
nWidth	Width of load rectangle
nHeight	Height of load rectangle

Description:

This function sets the load rectangle. This will cause an image loaded to be cropped. Portions of the image falling outside the coordinates of this rectangle will be discarded, and will not appear in the image bitmap of the DIB created by the load.

NOTE: To reset ImageGear to its default behavior of loading the whole image, set all parameters to 0.

NOTE: See also **IG_load_rect_get()** function.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
/* Set to only load the upper left 1000 x 500 of the image: */
IG_load_rect_set ( 0, 0, 1000, 500 );
```

IG_load_size_get

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_size_get ( LPAT_DIMENSION lpWidth,
LPAT_DIMENSION lpHeight);
```

Arguments:

lpWidth	Far pointer to variable of type AT_DIMENSION to receive the load size width
lpHeight	Far pointer to variable of type AT_DIMENSION to receive the load size height

Description:

This function obtains the load size dimensions.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

NOTE: To reset ImageGear so that it will default to normal resolution, set both parameters to 0 using **IG_load_size_set()**.

Example:

```
AT_DIMENSION nWid, nHi;   /* Will receive current load size
settings */
IG_load_size_get ( &nWid, &nHi );
```

IG_load_size_set

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_size_set (
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight
   );
```

Arguments:

nWidth	Load size width to set
nHeight	Load size height to set

Description:

This function sets the load size dimensions. These are the dimensions that the DIB image bitmap created by a load, are to have, regardless of the dimensions in the file from which the image is loaded. The effect is similar to loading and then re-sizing the image.

NOTE: To reset ImageGear so that it will default to loading the entire image, set both parameters to 0.

NOTE: See also **`IG_load_size_get()`** function.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
/* On loading, map the image into a 512 x 256 image bitmap: */
IG_load_size_set ( 512, 256 );
```

IG_load_tag_CB_register

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tag_CB_register ( LPFNIG_TAG_SET
lpfnTagSet, LPVOID lpPrivate );
```

Arguments:

lpfnTagSet	Far pointer to callback function to be called with each Tag encountered while loading
lpPrivate	Far pointer to private data (passed to callback function)

Description:

This function registers a callback function of type **LPFNIG_TAG_GET**, **LPFNIG_TAG_SET** or **LPFNIG_TAG_USER_GET** (See in the Core Component Callback Functions Reference section). These callback function types are defined by ImageGear to take a certain set of parameters and to return data to you. Your callback function must supply ImageGear with the type of data required by the callback that you choose.

Once you have written a callback function in one of the types listed above, this function should be called to register it. Once registered, your function will be called once for each Tag encountered while loading a file. A “tag” may also be known as an element in the image's header, or as non-image data. The TIFF format popularized the use of the word “tag”.

Different file formats have different sets of tags.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
LPFNIG_TAG_SET    MyTagGet;     /* Declare type of function */
{
static DWORD      dwPrivateFlags;
 …
IG_load_tag_CB_register ( MyTagSet, (LPVOID) &dwPrivateFlags );
```

IG_load_thumbnail

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_thumbnail ( const LPSTR
lpszFileName, LPHIGEAR lphIGear );
```

Arguments:

lpszFileName	Name of file
lphIGear	Far pointer to HIGEAR variable to receive handle

Description:

This function loads a thumbnail (if one exists in the file) from file **lpszFileName**, and returns the **HIGEAR** handle of the resulting image to the **HIGEAR** object pointed to by argument **lphIGear**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

See the example for function **IG_save_thumbnail_set()**.

IG_load_thumbnail_FD

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_thumbnail_FD (AT_INT fd, LONG
lOffset, LPHIGEAR lphIGear );
```

Arguments:

fd	File Descriptor handle
lOffset	Offset in file at which file image begins
lphIGear	Far pointer to HIGEAR variable to receive handle

Description:

This function loads a thumbnail from the open file indicated by **fd**. The handle of the resulting image is returned to the **HIGEAR** variable pointed to by argument **lphIGear**.

NOTE: See also functions **IG_load_thumbnail()**, and **IG_save_thumbnail_set()**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear1;     /* handle ret'd by IG_load_FD  */
HIGEAR  hIGear2;     /* handle for thumbnail  */
INT     fd;          /* DOS File Descriptor  */
LONG lOffset;              /* offset to image in file */
UINT nPageNum;             /* will be 0 for this call */
AT_ERRCOUNT nErrcount;     /* to test for errors  */
fd = _lopen ( "picture.bmp", OF_READ );    /* open file  */
nPageNum  = 0;          /* not a multi-page file  */
lOffset    = 0;         /* access file from start */
/* Load image, and obtain its HIGEAR handle:    */
nErrcount = IG_load_FD ( fd, lOffset, nPageNum, 0, &hIGear1 );
if ( nErrcount == 0 )
{ nErrcount = IG_load_thumbnail_FD ( fd, lOffset, &hIGear2 ); }
```

IG_load_thumbnail_mem

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_thumbnail_mem ( LPVOID lpImage,
AT_UINT nImageSize, LPHIGEAR lphIGear );
```

Arguments:

lpImage	Far pointer to start of file image in memory
nImageSize	Size of image in memory.
lphIGear	Far pointer to HIGEAR variable to receive handle

Description:

This function loads a thumbnail from the file image located in memory at location **lpImage**. The handle of the resulting image is returned to the **HIGEAR** variable pointed to by argument **lphIGear**.

NOTE: See also functions **IG_load_thumbnail()**, and **IG_save_thumbnail_set()**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear1;       /* handle ret'd by IG_load_mem  */
HIGEAR hIGear2;       /* ret'd by IG_load_thumbnail_mem  */
char far * lpWhereFile;    /* ptr to image file in mem   */
AT_UINT nImageSize;   /* Size of image in memory */
UINT nPageNum;      /* will be 0 for this call     */
AT_ERRCOUNT nErrcount;/* to test for errors         */
nPageNum     = 0;     /* not a multi-page file      */
lpWhereFile  = ...;   /* where mem image of file begins */
nImageSize = ...;   /* Size of image in memory */
nErrcount = IG_load_mem ( lpWhereFile, nImageSize, nPageNum, 0,
&hIGear1 );
if ( nErrcount == 0 )
   { nErrcount = IG_load_thumbnail_mem ( lpWhereFile,
nImageSize, &hIGear2 ); }
```

IG_load_tiles_stitch

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch(const LPSTR
lpszFileName, UINT nPage, LPAT_STITCH  lpStitch, LPHIGEAR
lphIGear);
```

Arguments:

lpszFileName	Set to the filename of the image to load and stitch
nPage	Set to the number of the page to load and stitch. If the file is not multipage, set this to 1.
lpStitch	Set to a structure of type AT_STITCH, which defines the reference tile number, and the number of row and columns of tiles
lphIGear	ImageGear returns you a HIGEAR handle to the newly stitched image in memory.

Description:

This function loads and stitches together a tiled image, returning you a **HIGEAR** handle to the image in memory.

As of this writing, ImageGear supports the following formats that sometimes use tiled storage: TIFF, IOCA, MO:DCA. To determine whether or not the image has tiles, call **IG_tile_count_get()**.

The AT_STITCH structure allows you to supply ImageGear with information on which tiles to use as the upper-left corner in the new stitched image, and how many tile rows and columns should be stitched together:

```
typedef struct tag AT_STITCH
{
   LONG uRefTile;   /* Upper left-hand corner tile #  */
   LONG uTileRows;  /* Number of tiles to stitch across */
   LONG uTileCols;  /* Number of tiles to stitch down   */
}AT_STITCH, FAR *LPAT_STITCH;
```

NOTE: For a graphical representation of how this works, see the section Working with Tiled Images Simply loading the file does not cause it to be displayed. Refer to **IG_dspl_image_draw()** and related routines, for how to display an image once it is in memory.

The **nPageNum** argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set **nPageNum** to 1 for a non-multi-page file.

NOTE: If you set nPageNum to < 1, ImageGear will default the value to 1; if you set nPageNum to greater than the number of pages in the document, ImageGear will default the value to the last page number.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

NOTE: See also **IG_load_tiles_stitch_FD()**, **IG_load_tiles_stitch_mem()**, **IG_tile_count_get()**functions.

For a complete discussion of working with tiled images see the section Working with Tiled Images.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
AT_STITCH lpStitch;
CHAR      szFile[_MAX_PATH]
```

```
nErrCount = IG_load_tiles_stitch(szFile, 1, lpStitch, &hIGear);
```

IG_load_tiles_stitch_FD

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_FD(AT_INT fd, LONG
lOffset, UINT nPage, LPAT_STITCH lpStitch, LPHIGEAR lphIGear);
```

Arguments:

fd	Set to the File Descriptor handle for the image you would like to load and stitch.
lOffset	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. `lOffset` is usually 0.
nPage	Set to the page number to load if this is a multipage file. Set to 1 if this is not a multipage file.
lpStitch	Set to a structure of type `AT_STITCH`, which defines the reference tile number, and the number of rows and columns of tiles.
lphIGear	Returns a `HIGEAR` handle to the newly stitched-together image

Description:

This function loads and stitches together a tiled image, returning you a **`HIGEAR`** handle to the image in memory. Unlike **`IG_load_tiles_stitch()`**, this function is used when the file is already open and you have its File Descriptor handle.

As of this writing, ImageGear supports the following formats that sometimes use tiled storage: TIFF, IOCA, MO:DCA. To find out whether the image has tiles, call **`IG_tile_count_get_FD()`**function.

The `AT_STITCH` structure allows you to tell ImageGear which tile to use as the upper-left corner in the new stitched image, and how many tile rows and columns should be stitched together:

```
typedef struct tag AT_STITCH
{
   LONG uRefTile;     /*Upper left-hand corner tile #  */
   LONG uTileRows;    /* Number of tiles to stitch across  */
   LONG uTileCols;/* Number of tiles to stitch down*/
}AT_STITCH, FAR *LPAT_STITCH;
```

NOTE: For a graphical representation of how this works, see the section Working with Tiled ImagesNote that simply loading and stitching the file does not cause it to be displayed. Refer to **`IG_dspl_image_draw()`** and related routines, for how to display an image once it is in memory.

NOTE: `lOffset` represents the number of bytes, positive or negative, from the position in the file currently pointed to by `fd`. The current `fd` position may have been moved around a few times so that it is no longer pointing to the beginning of the image's header information. Be sure to keep this in mind as you set the value of `lOffset`.

The **`nPageNum`** argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set **`nPageNum`** to 1 for a non multi-page file.

NOTE: If you set `nPageNum` to < 1, ImageGear will default the value to 1; if you set `nPageNum` to greater than the number of pages in the document, ImageGear will default the value to the last page number.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

NOTE: See also **`IG_load_tiles_stitch()`**, **`IG_load_tiles_stitch_mem()`**, **`IG_tile_count_get_FD()`**functions.

For a complete discussion of working with tiled images, see the section Working with Tiled Images.

NOTE: If you wish to make a subsequent call to **`IG_info_get_FD_ex()`**, you must first move the file pointer (of loaded image) to the beginning of the image's header information, or you will receive an error. This happens because after an image is loaded, the file pointer is positioned at the end of the image in the file.
To avoid the error:

- Call **`IG_load_FD()`**.

- Call the appropriate C function, that will set the pointer back to the beginning of the image's header information.
- Call **IG_info_get_FD_ex()**.

Example:

```
HIGEAR hIGear;      /* handle ret'd by IG_load_FD */
INT fd;          /* DOS File Descriptor  */
LONG lOffset;      /* offset to image in fill  */
UINT nPageNum;     /* will be 0 for this call  */
AT_ERRCOUNT nErrcount;   /* to test for errors  */
AT_STITCH stitchStruct;
LONG lHorizTileCount, lVertTileCount;
fd = _lopen ( "picture.tif", OF_READ );  /* open file  */
nPagenum  = 3;       /* load tiles from page 3  */
lOffset    = 0;         /* access file from start  */
stitchStruct.uRefTile = 5;     /* Use tile 5 as reference  */
stitchStruct.uTileRows = 2;     /* Stitch 2 tiles across  */
stitchStruct.uTileCols = 2;     /* Stitch 2 tiles down  */
/* Load tiles, stitch together and return HIGEAR handle:  */
nErrcount = IG_load_tiles_stitch_FD ( fd, lOffset, nPageNum,
&stitchStruct, &hIGear );
```

IG_load_tiles_stitch_mem

Group: Image Loading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_mem(LPVOID  lpImage,
AT_UINT nImageSize, UINT nPage, LPAT_STITCH lpStitch, LPHIGEAR
lphIGear);
```

Arguments:

lpImage	A far pointer to the memory location of an image file that is currently in memory
nImageSize	Size of image in memory.
nPage	Set to the page number to load if this is a multipage file. Set to 1 if this is not a multipage file.
lpStitch	Set to a structure of type AT_STITCH, which defines the reference tile number, and the number of row and columns of tiles
lphIGear	Returns a HIGEAR handle to the newly stitched-together image.

Description:

This function loads and stitches together a tiled image that has already been loaded into memory, returning you a **HIGEAR** handle to the image. The argument **lpImage** is a **FAR** pointer to the start of the image file in memory.

As of this writing, ImageGear supports the following formats that sometimes use tiled storage: TIFF, IOCA, MO:DCA. To find out whether the image has tiles, call **IG_tile_count_get_mem()** function.

The **AT_STITCH** structure allows you to tell ImageGear which tile to use as the upper-left corner in the new stitched image, and how many tile rows and columns should be stitched together:

```
typedef struct tag AT_STITCH
{
   LONG uRefTile;        /*Upper left-hand corner tile #  */
   LONG uTileRows;     /* Number of tiles to stitch across */
   LONG uTileCols;     /* Number of tiles to stitch down   */
}AT_STITCH, FAR *LPAT_STITCH;
```

For graphical representation of how this works, see the section Working with Tiled Images.

NOTE: Simply loading and stitching the file does not cause it to be displayed. Refer to **IG_dspl_image_draw()** and related routines, for how to display an image once it is in memory. See also **IG_load_file_display()** function.

The **nPageNum** argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set **nPageNum** to 1 for a non multi-page file.

NOTE: If you set nPageNum to < 1, ImageGear will default the value to 1; if you set nPageNum to greater than the number of pages in the document, ImageGear will default the value to the last page number.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

NOTE: See also **IG_load_tiles_stitch()**, **IG_load_tiles_stitch_FD()**, **IG_tile_count_get_mem()** functions.

For a complete discussion of working with tiled images, see the section Working with Tiled Images.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
char far *lpWherFile;
AT_UINT nWholeSize;
UINT   nPageNum;
AT_STITCH stitchStruct;
nErrcount = IG_load_tiles_stitch_mem(lpWhereFile, nWholeSize,
nPageNum, &stitchStruct, &hIGear);
```

IG_LUT_copy

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_copy(HIGLUT SrcLUT, HIGLUT
*lpDstLUT);
```

Arguments:

SrcLUT	Source LUT handle.
lpDstLUT	New LUT handle.

Description:

This function copies the LUT to a new HIGLUT object

Bits Per Pixel: 8...16-bit grayscale

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

IG_LUT_copy_to_byte_array

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_copy_to_byte_array(
    HIGLUT SrcLUT,
    AT_INT ArrayLength,
    AT_BYTE *lpArray
    );
```

Arguments:

SrcLUT	LUT to be copied.
ArrayLength	Array length.
lpArray	Byte array.

Description:

This function copies LUT a to byte array. Array length must be equal to **2^InputDepth**, where **InputDepth** is the input depth of the LUT.

Bits Per Pixel: 8...16-bit grayscale

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

None

IG_LUT_copy_to_word_array

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_copy_to_word_array(
```

```
HIGLUT SrcLUT,
AT_INT ArrayLength,
AT_WORD *lpArray
);
```

Arguments:

SrcLUT	LUT to be copied.
ArrayLength	Array length.
lpArray	Word array.

Description:

This function copies LUT a to word array.

Array length must be equal to **`2^InputDepth`**, where **`InputDepth`** is the input depth of the LUT.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

None

IG_LUT_create

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_create(
   AT_INT InputDepth,
   AT_BOOL InputIsSigned,
   AT_INT OutputDepth,
   AT_BOOL OutputIsSigned,
   HIGLUT *lpLUT
   );
```

Arguments:

InputDepth	Input bit count of the LUT.
InputIsSigned	Shows whether input of the LUT is signed.
OutputDepth	Output bit count of the LUT.
OutputIsSigned	Shows whether output of the LUT is signed.
lpLUT	Handle of the created LUT object.

Description:

This function creates a LUT object.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

IG_LUT_destroy

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_destroy(
HIGLUT lut
);
```

Arguments:

lut	Handle of the LUT to be destroyed.

Description:

This function destroys a LUT object.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

IG_LUT_input_depth_get

Group: Grayscale LUT

Declaration:

```
AT_INT ACCUAPI IG_LUT_input_depth_get(HIGLUT lut);
```

Arguments:

lut	LUT handle.

Description:

This function returns input bit count of the LUT.

Input depth defines the length (number of entries) of the lut, as follows:

LUTLength = 2^InputDepth.

Bits Per Pixel: 8...16-bit

Return Value:

Input bit count of the LUT.

None

IG_LUT_input_is_signed_get

Group: Grayscale LUT

Declaration:

```
AT_BOOL ACCUAPI IG_LUT_input_is_signed_get(HIGLUT lut);
```

Arguments:

lut	LUT handle.

Description:

This function tells whether input of the LUT is signed.

Bits Per Pixel: 8...16-bit

Return Value:

`TRUE` if the input is signed, `FALSE` - otherwise.

None

IG_LUT_is_valid

Group: Grayscale LUT

Declaration:

```
AT_BOOL ACCUAPI IG_LUT_is_valid(
HIGLUT lut
);
```

Arguments:

lut	LUT handle.

Description:

This function checks whether HIGLUT is valid.

Bits Per Pixel: 8...16-bit

Return Value:

`TRUE` if `HIGLUT` is valid, `FALSE` - otherwise.

None

IG_LUT_item_get

Group: Grayscale LUT

Declaration:

```
AT_INT ACCUAPI IG_LUT_item_get(HIGLUT lut, AT_INT32 index);
```

Arguments:

lut	Handle of the LUT.
index	Index of the item to be returned.

Description:

This function returns LUT item.

If LUT input is unsigned, valid values for index are in range [0, 2^InputDepth-1].

If LUT input is signed, valid values for index are in range [-2^(InputDepth-1), 2^(InputDepth-1)-1].

Bits Per Pixel: 8...16-bit

Return Value:

LUT item.

None

IG_LUT_item_set

Group: Grayscale LUT

Declaration:

```
AT_VOID ACCUAPI IG_LUT_item_set(HIGLUT lut, AT_INT32 index,
AT_INT value);
```

Arguments:

lut	LUT handle.
index	Index of the item.
value	Value of the item.

Description:

This function sets the LUT item.

If LUT input is unsigned, valid values for index are in range [0, 2^InputDepth-1].

If LUT input is signed, valid values for index are in range [-2^(InputDepth-1), 2^(InputDepth-1)-1].

Bits Per Pixel: 8...16-bit

Return Value:

None.

None

IG_LUT_length_get

Group: Grayscale LUT

Declaration:

```
AT_INT ACCUAPI IG_LUT_length_get(
HIGLUT hlut
);
```

Arguments:

hlut	LUT handle.

Description:

This function returns the number of entries in the LUT.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of entries in the LUT.

IG_LUT_output_depth_get

Group: Grayscale LUT

Declaration:

```
AT_INT ACCUAPI IG_LUT_output_depth_get(HIGLUT lut);
```

Arguments:

lut	LUT handle.

Description:

This function returns output bit count of the LUT.

Output depth defines the size of each entry of the LUT.

Bits Per Pixel: 8...16-bit

Return Value:

Returns output bit count of the LUT.

None

IG_LUT_output_is_signed_get

Group: Grayscale LUT

Declaration:

```
AT_BOOL ACCUAPI IG_LUT_output_is_signed_get(HIGLUT lut);
```

Arguments:

lut	LUT handle.

Description:

This function tells whether output of the LUT is signed.

Bits Per Pixel: 8...16-bit

Return Value:

TRUE - output of the LUT is signed, FALSE - otherwise.

None

IG_LUT_size_get

Group: Grayscale LUT

Declaration:

```
AT_INT ACCUAPI IG_LUT_size_get(
HIGLUT hlut
);
```

Arguments:

hlut	Handle of the LUT.

Description:

This function returns the size, in bytes, of the LUT data.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the size, in bytes, of the LUT data.

None

IG_LUT_update_from_byte_array

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_update_from_byte_array(
HIGLUT lut,
AT_INT ArrayLength,
const AT_BYTE *lpArray
);
```

Arguments:

lut	LUT handle.
ArrayLength	Array length.
lpArray	Byte array.

Description:

This function updates LUT from a byte array.

Array length must be equal to **`2^InputDepth`**, where **`InputDepth`** is the input depth of the lut.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

None

IG_LUT_update_from_word_array

Group: Grayscale LUT

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_update_from_word_array(
HIGLUT lut,
AT_INT ArrayLength
const AT_WORD *lpArray
);
```

Arguments:

lut	LUT handle.
ArrayLength	Number of WORDs in the array
lpArray	Word array.

Description:

This function updates LUT from a word array.

Array length must be equal to **2^InputDepth**, where **InputDepth** is the input depth of the LUT.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

None

IG_mem_CB_register

Group: Callback

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mem_CB_register(LPFNIG_MEM_ALLOC
lpfnAllocFunc, LPFNIG_MEM_REALLOC lpfnReAllocFunc,
LPFNIG_MEM_FREE lpfnFreeFunc);
```

Arguments:

lpfnAllocFunc	Far pointer to your function to be called for memory allocations.
lpfnReAllocFunc	Far pointer to your function to be called for memory reallocations.
lpfnFreeFunc	Far pointer to your function to be called to free memory free.

Description:

This function registers your own callback functions to be called to do large memory allocations, memory reallocations, and memory freeing. As shown in the prototype, your memory callback functions must be of types **LPFNIG_MEM_ALLOC**, **LPFNIG_MEM_REALLOC** and **LPFNIG_MEM_FREE**.

NOTE: Your memory allocation functions will only be used when large allocations (allocations greater than 1024) are performed.

Set any of the three arguments to **NULL** if you want ImageGear to use its own definition for these functions.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
/* Memory Alloc callback function definition
*/
LPBYTE  ACCUAPI MyMemAlloc(DWORD dwSize)/* number of bytes to
alloc    */
{
   /* Put your own memory allocation code here */
   return( buffer);
};
/* Memory ReAlloc callback function definition
*/
LPBYTE ACCUAPI MyMemReAlloc( LPBYTE lpBuffer, DWORD
   dwSize)
{
   /* Put your own memory reallocation code here */
   return( lpBuffer);
};
/* Memory Free callback function definition   */
LPBYTE  ACCUAPI MyMemFree(LPBYTE lpBuffer)
{
   /*Put your free-the-memory code here */
   return NULL;
```

```
};
/* Registration Example  */
/* Example one  */
/* Register your own callback functions for all memory routines
*/
nErrcount = IG_mem_CB_register(MyMemAlloc, MyMemReAlloc,
MyMemFree);
/* Example two */
/* Supply callbacks for memory alloc only  */
nErrcount = IG_mem_CB_register(MyMemAlloc, NULL, NULL);
```

IG_mpf_info_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_info_get(
[IN] HMIGEAR hMIGear,
[OUT] LPAT_MODE lpFileType,
   );
```

Arguments:

hMIGear	Handle to allocated a multi-page image.
lpFileType	Far pointer indicating where to receive file types such as `IG_FORMAT_TIF`, or `IG_FORMAT_MODCA`. For a complete list of format types, see the `accucnst.h` file.

Description:

If a multi-page image `hMIGear` is associated with the external file image, then this function returns the file format type of the external image through a second parameter, `lpFileType`. This constant is from the format constant list, which is defined in the `accucnst.h` include file. If it is not associated with the file, then this function returns an error.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;  /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS;  /* will hold returned
error count  */
AT_MODE nFileType;
 …
```

```
/* initialize multi-page image and assign it with external file
*/

   iErrCnt = IG_mpf_info_get( hMIGear, &nFileType );

   if (!iErrCnt)
   printf("File type:%i\n", (INT)nFileType);
```

IG_mpf_page_count_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_page_count_get(
[IN]HMIGEAR hMIGear,
[OUT]LPUINT lpnPageCount
 )
```

Arguments:

hMIGear	Handle of the allocated multi-page image.
lpnPageCount	Pointer indicating where to return number of pages of the associated file

Description:

Uses the **IG_mpi_file_open()** function to get the number of pages in the external file if it is associated with multi-page image. If the file is not associated with a multi-page image file, the value is 0.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image  */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCountI = 0; /* number of pages that should get from
multi-page image */
UINT   nPageCountF = 0; /* number of pages that should get from
external source */
HIGEAR hIGear;      /* handle of an image */

 …
```

```
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCountI );
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCountF );

printf("Number of pages of multi-page image is:%i\n",
nPageCountI);
printf("Number of pages of external source is:%i", nPageCountF);
```

IG_mpf_page_delete

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_page_delete(
[IN]HMIGEAR hMIGear,
[IN]UINT  nStartPage,
[IN]UINT  nCount
 )
```

Arguments:

hMIGear	Handle to the allocated multi-page image.
nStartPage	The first page to be deleted
nCount	Number of pages to delete

Description:

This function deletes the `nCount` number of pages starting with the `nStartPage`. It then shifts pages with higher numbers to fill the space in the external file associated with the multi-page image using function **`IG_mpi_file_open()`**. The multi-page image itself is not changed. Either the multi-page image, or in the external file pages are numbered starting with 0.

NOTE: Not all format filters are supported by this operation. Use the **`IG_fltr_info_get()`** function to obtain all information about the supported features for a particular format filter.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count  */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i;
HIGEAR hIGear;      /* handle of an image */


 …
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
   if (!nErrCount)
      if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
         nErrCount = IG_mpf_page_delete( hMIGear, i, 1 );
```

IG_mpf_page_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_page_get(
[IN]HMIGEAR hMIGear,
[IN]UINT  nPage,
[OUT]LPHIGEAR lphIGear
 )
```

Arguments:

hMIGear	Handle to the allocated multi-page image.
nPage	The index page to load
lphIGear	Indicates where to return the image handle

Description:

If a multi-page image is associated with the external file, it loads a page of the specified index from an external file and returns it using the parameter `lphIGear`. If a multi-page image is not associated with the external file, or a failure to load a page occurs, then an error is set. This function does not change the multi-page image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear; /* handle to multi-page image */
AT_ERRCOUNT  nErrCount = IGE_SUCCESS; /* will hold returned
error count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i;
HIGEAR hIGear;    /* handle of an image */
/* initialize multi-page image and assign it with external file
*/
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
   if (!nErrCount)
   if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
   {
   nErrCount = IG_mpf_page_load( hMIGear, i, &hIGear );
    …
   if( IG_image_is_valid( hIGear ) )
   IG_image_delete( hIGear );
   }
```

IG_mpf_page_info_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_info_get(
[IN]HMIGEAR hMIGear,
[IN]UINT  nPage,
[OUT] LPAT_MODE lpCompression,
[OUT]LPAT_DIB lpDib
)
```

Arguments:

hMIGear	The handle to allocated the multi-page image.
nPage	Index of the page, starting with 0.
lpCompression	Far pointer indicating where to receive the compression method of the given image. It is used for constants such as `IG_COMPRESSION_NONE`, or `IG_COMPRESSION_JPEG`. A complete list of compression methods can be found in the `accucnst.h` file.
lpDib	The far pointer indicating where to return a DIB associated with the page.

Description:

If multi-page image `hMIGear` is associated with an external file, then this function returns information about the page with an `nPage` index from the external file. Pages are numbered starting with 0. If the image is not associated with an external file, then this functions returns an error.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;   /* handle to multi-page image  */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCount = 0;
UINT   i;
AT_MODE    nCompression;
AT_DIB   Dib;
char   str[80];
 …
/* initialize multi-page image and assign it with external file
*/
     nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );

   for( i = 0; i < nPageCount; i++)
     if (!nErrCnt)
   {
   nErrCnt = IG_mpf_page_info_get( hMIGear, i, &nCompression,
&Dib );
   if (!nErrCnt)
   {
   IG_guidlg_compression_name_get( nCompression, str,
sizeof(str) );
   printf( "Page %i\n  Compression method: %s\n", i, str );
   }
```

```
}
```

IG_mpf_page_info_get_ex

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_page_info_get_ex(
HMIGEAR hMIGear,
UINT nPage,
LPAT_MODE lpCompression,
HIGDIBINFO* lpDIB
);
```

Arguments:

hMIGear	Handle to allocated multi-page image.
nPage	Index of the page, starting with 0.
lpCompression	Pointer indicating where to receive the compression method of the given image. It is used for constants such as `IG_COMPRESSION_NONE`, or `IG_COMPRESSION_JPEG`. A complete list of compression methods can be found in the `accucnst.h` file.
lpDIB	Pointer indicating where to return a DIB info handle associated with the page.

Description:

If multi-page image `hMIGear` is associated with an external file, then this function returns information about the page with an `nPage` index from the external file. Pages are numbered starting with 0. If the image is not associated with an external file, then this function returns an error.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
HMIGEAR hMIGear;   /* handle to multi-page image  */
AT_ERRCOUNT nErrCount; /* will hold returned error count */
UINT nPageCount = 0;
UINT i;
AT_MODE nCompression;
HIGDIBINFO hDIB;
char str[80];
```

```
...
/* initialize multi-page image, assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for( i = 0; i < nPageCount; i++)
if (!nErrCnt)
{
   nErrCnt = IG_mpf_page_info_get_ex( hMIGear, i,
&nCompression, &hDIB );
   if (!nErrCnt)
   {
      IG_guidlg_compression_name_get( nCompression, str,
sizeof(str) );
      printf( "Page %i\n  Compression method: %s\n", i, str );
   }
}
```

IG_mpf_page_load

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_page_load(
[IN]HMIGEAR hMIGear,
[IN]UINT  nStartPage,
[IN]UINT  nCount
 )
```

Arguments:

hMIGear	The handle of the allocated multi-page image.
nStartPage	The first page to be loaded
nCount	The total number of pages to be loaded

Description:

If a multi-page image is associated with an external file, it loads and stores the specified number of pages from the external file into a multi-page image. To access the loaded pages, use the **IG_mpi_page_get()** function.

If the multi-page image is not associated with an external file, or a failure to load a page occurs, then an error is set. This function loads each Nth page from a file into the correspondent Nth page into the multi-page image. Previous page values are not deleted with function **IG_image_delete()**. If necessary, the number of pages is expanded to fit all loaded pages.

NOTE: The access to the same PDF document from multiple threads is not permitted because the multiple threads cannot share Adobe PDF Library datatypes. PDF doc created/opened in the main thread can be only used from the main thread.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i;
HIGEAR hIGear;     /* handle of an image */
 …
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );

for ( i = 0; i < nPageCount; i++ )
   if (!nErrCount)
   if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
   nErrCount = IG_mpf_page_load( hMIGear, i, 1 );
```

IG_mpf_page_save

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_page_save(
[IN]HMIGEAR hMIGear,
[IN]UINT  nStartPage,
[IN]UINT  nCount,
[IN]AT_MODEnCompression,
[IN]AT_MODE nSaveMode
 )
```

Arguments:

hMIGear	The handle of the multi-page image.
nStartPage	First page to save
nCount	Total number of pages to save
nCompression	Compression method to be used when saving the pages
nSaveMode	The methods for saving the pages can be: ■ IG_MPF_SAVE_INSERT ■ IG_MPF_SAVE_REPLACE

Description:

If the multi-page image is associated with an external file then this function saves the specified pages from multi-page image to the external file. If nSaveMode is IG_MPF_SAVE_INSERT, then the specified nCount number of pages are inserted, starting with nStartPage index, and all previous pages are shifted to a higher page number. If nSaveMode is IG_MPF_SAVE_REPLACE, then the function replaces the specified nCount number of pages, starting with the nStartPage index. If a page is NULL in the multi-page image, it is skipped and not saved.

This function takes each specified Nth page from a multi-page image and saves it as the Nth page into an external file. If the image is not associated with an external file, then it returns an error.

NOTE: Not all format filters support IG_MPF_SAVE_REPLACE mode. Use the **IG_fltr_info_get()** function to obtain information about the supported features for specific format filters.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count  */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i;
HIGEAR hIGear;      /* handle of an image */
 …
/* initialize multi-page image and assign it with external file
*/
```

```
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );

if ( !nErrCount && (nPageCount > 0) )
IG_mpf_page_save( hMIGear, 0, 1, IG_COMPRESSION_NONE,
IG_MPF_SAVE_INSERT );
```

IG_mpf_page_swap

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_page_swap(
[IN]   HMIGEAR hMIGear,
[IN]   UINT   nPage1,
[IN]   UINT   nPage2
 )
```

Arguments:

hMIGear	The handle of the multi-page image.
nPage1	The number of first page to swap.
nPage2	The number of second page to swap.

Description:

If the multi-page image is associated with an external file, then this function swaps pages with the given page numbers in the external file. If the multi-page image is not associated with an external file, an error is returned. Pages are numerated starting with 0. The multi-page image is not changed.

NOTE: Not all format filters support **`IG_mpf_page_swap()`** mode. Use the **`IG_fltr_info_get()`** function to obtain information about the supported features for specific format filters.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image  */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
```

```
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */


 …
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );

/* Swap first page and last pages */
if (!nErrCount && nPageCount > 1)
   nErrCount += IG_mpf_page_swap(hMIGear, 0, nPageCount - 1 );
```

IG_mpf_page_unload

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_page_unload(
[IN]   HMIGEAR hMIGear,
[IN]   UINT   nStartPage,
[IN]   UINT   nCount
 )
```

Arguments:

hMIGear	The handle to the valid multi-page image.
nStartPage	The number of the first page to unload.
nCount	The total number of pages to unload

Description:

This function calls **`IG_image_delete()`** for the `nCount` number of pages in the multi-page image, starting with the `nStartPage` position. The number of pages in the multi-page image is not changed, but the specified positions are removed and set to a default value of `NULL`. The pages are numbered starting with 0.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
```

```
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i;
HIGEAR hIGear;  /* handle of an image */


 …
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );

for ( i = 0; i < nPageCount; i++ )
   if ( IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
   nErrCount = IG_mpf_page_unload( hMIGear, i, 1 );
```

IG_mpf_tile_count_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT IG_mpf_tile_count_get(
[IN]   HMIGEAR hMIGear,
[IN]   UINT   nPage,
[OUT]  LPUINT lpTileRows,
[OUT]  LPUINT lpTileCols
 )
```

Arguments:

hMIGear	The handle of the multi-page image
nPage	The page number from which to get the tile count
LpTileRows	The far pointer to a `UINT` variable to receive the number of tile rows (for tiled images)
LpTileCols	The far pointer to a `UINT` variable to receive the number of tile columns (tiled images)

Description:

If the multi-page image is associated with an external file, then this function returns the tile information for the specified page number. An error returns if the image is not associated with an external file. Pages are numbered starting with 0.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   nTileRows;
UINT   nTileCols;


 …
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );

/* Count number of tiles in the first page */
if (!nErrCount && nPageCount > 0)
   nErrCount += IG_mpf_tile_count_get(hMIGear, 0, &nTileRows,
&nTileCols );

if (!nErrCount)
{
   printf( "Number of tiles in a row:%i\n", nTileRows );
   printf( "Number of tiles in a colomns:&i\n", nTileCols );
}
```

IG_mpi_CB_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_CB_get(
[IN]   HMIGEAR hMIGear,
[IN]   DWORD dwCBID,
[OUT]  LPVOID FAR* lplpPrivate,
[OUT]  LPFNIG_MPCB_UPDATE FAR* lplpfnUpdate
)
```

Arguments:

hMIGear	The handle of the allocated multi-page image.
dwCBID	A unique identifier of the private data and function.
lplpPrivate	A pointer indicating where to receive the private data.
lplpfnUpdate	A pointer indicating where to receive the update function.

Description:

See the **IG_mpi_CB_set()** documentation for a description of how notification works with multi-page images. This function returns information about the associated private data pointer and update function using the `dwCBID` identifier.

If there is no association with an external file, then `NULL` values are assigned to both pointers.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
LPVOID lpData;
LPFNIG_MPCB_UPDATE        lpUpdateFunc;
DWORD  dwCBID;
 …
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear,
_MPWndUpdate, &dwCBID );
 …
nErrCount = IG_mpi_CB_get( hMIGear, dwCBID, &lpData,
&lpUpdateFunc );
 …
VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
     …
       break;
```

```
        case  IG_MPCBMODE_MPI_ASSOCIATED:
          …
            break;
        case  IG_MPCBMODE_MPI_CLOSE:
          …
            break;
          …
        }
    }
```

IG_mpi_CB_reset

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_CB_reset(
[IN]   HMIGEAR hMIGear,
[IN]   DWORDdwCBID
)
```

Arguments:

hMIGear	Handle to allocated multi-page image.
dwCBID	Unique identifier of private data and function.

Description:

See the **IG_mpi_CB_set()** documentation for a description of how notification works with multi-page images. This function removes previously associated callback data from the multi-page image using the `dwCBID` identifier. After calling this function, the appropriate callback function receives notifications and removes the data from the active list.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count  */
LPVOID lpData;
LPFNIG_MPCB_UPDATE    lpUpdateFunc;
DWORD  dwCBID;
 …
```

```
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear,
_MPWndUpdate, &dwCBID );
 …
nErrCount = IG_mpi_CB_reset( hMIGear, dwCBID );
 …
}
VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID        lpPrivate,     /* Private data passed in
*/
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
     …
       break;
   case  IG_MPCBMODE_MPI_ASSOCIATED:
     …
       break;
   case  IG_MPCBMODE_MPI_CLOSE:
     …
       break;
     …
   }
}
```

IG_mpi_CB_reset_all

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_CB_reset_all(
[IN]   HMIGEAR   hMIGear
)
```

Arguments:

hMIGear	The handle to the allocated multi-page image.

Description:

This function works the same way as the **IG_mpi_CB_reset()** function, but removes all callback data and functions from the multi-page image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count  */
DWORD  dwCBID1, dwCBID2;

 …
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear,
_MPWndUpdate, &dwCBID1 );
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear,
_MPWndUpdate, &dwCBID2 );
 …
nErrCount = IG_mpi_CB_reset_all( hMIGear );
 …
}

VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in  */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
     …
       break;
   case  IG_MPCBMODE_MPI_ASSOCIATED:
     …
       break;
   case  IG_MPCBMODE_MPI_CLOSE:
     …
       break;
     …
   }
}
```

IG_mpi_CB_set

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_mpi_CB_set(
[IN]   HMIGEAR hMIGear,
[IN]   LPVOID lpPrivate,
[IN]   LPFNIG_MPCB_UPDATE lpfnUpdate,
[OUT]  LPDWORD lpdwCBID,
)
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
lpPrivate	Any private data.
lpfnUpdate	The pointer to the update function
lpdwCBID	Indicates where to return the ID of the associated callback data.

Description:

Multi-page images allow you to notify the application about status changes. Use this function to call code that associates the given multi-page image `hMIGear` with any `lpPrivate` data, and updates the defined function as shown below:

```
typedef  VOID (LPACCUAPI LPFNIG_MPCB_UPDATE)(
   DWORD          dwCBID,
   LPVOID        lpPrivate,     /* Private data passed in
*/
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
);
```

It returns a unique identifier, which allows multi-page associations with the given information. After the association is made, the application receives notifications about changes to the multi-page image through this function.

The Callback function parameters are described in the table below:

dwCBID	The identifier allocated for this callback function by **IG_mpi_CB_set()**
lpPrivate	The private data associated with this identifier
nMode	The type of multi-page image change
nPage, nCount	Usually the first number and and total number of changed pages.

The following table lists all possible values for `nMode` and the appropriate sense of the `nPage` and `nCount` parameters:

nMode	nPage	nCount	Description
IG_MPCBMODE_MPI_DELETE	Not used	Not used	Notifies the application that the multi-page image is going to be deleted.
IG_MPCBMODE_MPI_ASSOCIATED	Not used	Not used	Notifies the application that the multi-page image is associated with an external file.
IG_MPCBMODE_MPI_CLOSE	No used	Not used	Notifies the application that the multi-page image is going to close the associated external file
IG_MPCBMODE_MPI_CB_SET	Not used	Not used	Notifies the application that this callback data is set. This notification receives only the callback function that has just been set.
IG_MPCBMODE_MPI_CB_RESET	Not used	Not used	Notifies the application that this callback data is to be reset.
IG_MPCBMODE_MPI_PAGEINSERTED	index of where new pages starts	number of new pages inserted	Notifies the application that new pages have been inserted into the multi-page image.
IG_MPCBMODE_MPI_PAGEUPDATED	index of the first updated page	number of updated pages starting from `nPage`	Indicates that the application has updated pages in the multi-page image
IG_MPCBMODE_MPI_PAGEDELETED	index of first deleted page	number of deleted pages	Indicates that the application has deleted pages in the multi-page image.
IG_MPCBMODE_MPF_PAGEINSERTED	index of where new pages starts	number of new pages inserted	Indicates that the application has inserted new pages into the external file image.
IG_MPCBMODE_MPF_PAGEUPDATED	index of the first updated page	number of updated pages starting from `nPage`	Indicates that the application has updated pages in the external multi-page image file.

nMode	nPage	nCount	Description
IG_MPCBMODE_MPF_ PAGEDELETED	index of the first deleted page	number of deleted pages	Application deleted pages in the external multi-page image file

NOTE: The notification function blocks the execution of the operation that performed the action. This function can be used for thread synchronization. We do not recommended that you call a multi-page API from the notification function (to prevent an unlimited loop from occurring).

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEARhMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
LPVOID lpData;
LPFNIG_MPCB_UPDATE    lpUpdateFunc;
DWORD  dwCBID;


 …
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear,
_MPWndUpdate, &dwCBID );
 …
nErrCount = IG_mpi_CB_get( hMIGear, dwCBID, &lpData,
&lpUpdateFunc );
 …
}

VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
     …
       break;
```

```
        case  IG_MPCBMODE_MPI_ASSOCIATED:
          …
            break;
        case  IG_MPCBMODE_MPI_CLOSE:
          …
            break;
          …
        }
    }
```

IG_mpi_close

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_close( HMIGEAR hMIGear );
```

Arguments:

hMIGear	The handle to the allocated multi-page image.

Description:

If the multi-page image `hMIGear` was previously associated with an external image file using the function **`IG_mpi_file_open()`**, then this function closes the file and frees all corresponding resources. If the multi-page image is not associated with an external file, then this function does nothing.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS;  /* will hold returned
error count */

nErrCount = IG_mpi_create( &hMIGear, 0 );
 …
/* any operations with hMIGear */

nErrCount = IG_mpi_close( hMIGear );
```

IG_mpi_create

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_create( LPHMIGEAR lphMIGear, UINT
nPages )
```

Arguments:

lpHMIGear	A pointer indicating where to return the handle of the allocated and initialized multi-page image.
nPage	The number of multi-page image pages that should be created.

Description:

This function allocates and initializes a new multi-page image and returns its handle. This new image is set with `nPages`. Each image is initialized with the default value `NULL`. If there is a failure, then the returned handle is `NULL` and an error is set.

The multi-page image is array of pages where each page is a `HIGEAR` object. All pages are numbered beginning with a **0** index, so that **0** - is the first page, **1** - is the second page, etc. If the image contains `nCount` number of pages, then its pages can be accessed through indexes 0 - `nCount`-1. The value of each page can be either `NULL` (default value) or value `HIGEAR` image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR       hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT   nErrCount = IGE_SUCCESS; /* will hold returned
error count */

nErrCount = IG_mpi_create( &hMIGear, 0 );
 …
/* any operations with hMIGear */

nErrCount = IG_mpi_delete( hMIGear );
```

IG_mpi_delete

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_mpi_delete( HMIGEAR hMIGear );
```

Arguments:

hMIGear	Handle to allocated multi-page image.

Description:

This function calls **IG_image_delete()** for all valid pages of multi-page image and frees all resources allocated with multi-page image handle. If it is associated with external file then it is closed with **IG_mpi_close()** before deletion.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */

nErrCount = IG_mpi_create( &hMIGear, 0 );
 …
/* any operations with hMIGear */

nErrCount = IG_mpi_delete( hMIGear );
```

IG_mpi_file_open

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_file_open(
[IN]   const LPCHAR lpFileName,
[IN]   HMIGEAR hMIGear,
[IN]   AT_MODE nFormat,
[IN ]  AT_MODE nOpenMode
)
```

Arguments:

lpFileName	The far pointer to the filename (you may include the path with the filename) of the image file to be associated with a given multi-page image
hMIGear	The handle to the allocated multi-page image.
nFormat	The format of the file, such as `IG_FORMAT_UNKNOWN` or `IG_FORMAT_TIF`. See the `accucnst.h` file for a list of all available `IG_FORMAT_`… constants.
nOpenMode	An `AT_MODE` constant, such as `IG_MP_OPENMODE_READONLY` or `IG_MP_OPENMODE_READWRITE`

Description:

This function allows you to associate a multi-page image with an external image file. After the association is made, you can then use different page manipulation functions, such as page load, save, delete, and swap. With this association operation, ImageGear allows you to store internal data, allowing you to make page operations faster than if using `IG_fltr_`… functions. This file can be opened with two modes - `IG_MP_OPENMODE_READONLY` and `IG_MP_OPENMODE_READWRITE`.

The first mode, read-only access, it is used only when page loading is necessary. It does not allow you to change the external file. When the image is opened with read only access, it sets the number of pages in the multi-page image equal to the number of pages in the external file using **`IG_mpi_page_count_set()`**function.

The second mode opens file for read-write access and allows all possible page operations with the external file. The multi-page image is not changed.

NOTE: Not all filters support all page manipulation operations. Use the function **`IG_fltr_info_get()`**, which returns the information about all supported features of a particular filter.

`IG_MP_OPENMODE_NONE` is also accepted as a value for `nOpenMode` and, in this case, this call is equivalent to the **`IG_mpi_close()`** call.

The `nFormat` parameter is used only if a new image file is to be created and `nOpenMode= IG_MP_OPENMODE_READWRITE.` In this case, the file of the specified format is created. In all other cases this parameter is ignored.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
```

```
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */

nErrCount = IG_mpi_create( &hMIGear, 0 );

if (!nErrCount)
nErrCount = IG_mpi_file_open( "picture1.tif", hMIGear, 0,
IG_MP_OPEN_READ );
…
nErrCount = IG_mpi_delete( hMIGear );
```

IG_mpi_file_save

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_mpi_file_save(
[IN]const LPCHAR lpFileName,
[IN]HMIGEAR hMIGear,
[IN]UINT nStartPageFile,
[IN]UINT nStartIndexDoc,
[IN]UINT nPageCount,
[IN]AT_MODE nFormat,
[IN]AT_MODE nSaveMode
   );
```

Arguments:

lpFileName	Output file name
hMIGear	Multi-page vector document
nStartPageFile	The page number in lpFileName after which pages from hMIGear are inserted. The first page is 0.
nStartIndexDoc	The page number of the first page in hMIGear to save.The first page is 0.
nPageCount	Total number of pages to save
nFormat	IG_FORMAT_PDF or IG_FORMAT_POSTSCRIPT
nSaveMode	IG_MPI_SAVE_OVERWRITE or IG_MPI_SAVE_APPEND

Description:

Use this function to save a multi-page vector document to a file.

NOTE: This function is only used for multi-page vector documents. The following formats are currently supported by this API: PDF, PostScript.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */

nErrCount = IG_mpi_create( &hMIGear, 0 );

if (!nErrCount)
nErrCount = IG_mpi_file_open( "picture1.tif", hMIGear, 0,
IG_MP_OPEN_READ );
…
nErrCount = IG_mpi_delete( hMIGear );
```

IG_mpi_info_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_info_get(
[IN]   HMIGEAR hMIGear,
[IN]   AT_MODE nMode,
[IN/OUT]LPVOID lpData,
[IN]   DWORD  dwSize
)
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
nMode	This argument is used to determine the type of information being retrieved
lpData	The far pointer to the buffer indicating where to return the allocated data
dwSize	The size of the buffer

Description:

This function returns status information for the multi-page image, such as the association type, open mode, as well as others. (*See the table below.*)

Value of parameters depends on `nMode`. Below in the table is list of all possible combinations.

AT_MODE constants	Type of third argument	dwSize	Description
`IG_MP_ASSOCIATION_TYPE`	`LPAT_MODE`	Sizeof (`AT_MODE`)	Returns the association type of the given multi-page image, which can be: ■ **`IG_MP_ASSOCIATE_NONE`** ■ **`IG_MP_ASSOCIATE_FILE`** ■ **`[*]IG_MP_ASSOCIATE_MEMORY`**
`IG_MP_OPENMODE`	`LPAT_MODE`	Sizeof(`AT_MODE`)	Returns the open mode of the associated file, which can be: ■ **`IG_MP_OPENMODE_NONE`** ■ **`IG_MP_OPENMODE_READONLY`** ■ **`IG_MP_OPENMODE_READWRITE`**
`IG_MP_FILE_NAME`	`LPCHAR`	Length of the buffer including last 0 byte	Returns the name of the associated file
`[*]IG_MP_MEM_BUFFER_PTR`	`LPBYTE`	4	Returns a pointer to the memory associated with the multipage image
`[*]IG_MP_MEM_BUFFER_SIZE`	`LPDWORD`	4	Returns the size of the associated memory
`IG_MP_FORMAT`			File format of the multi-page document. One of the `enumIGFormats` values.
`IG_MP_DOCUMENT`			Native document associated with the multi-page document.

[*] - not implemented now.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS;  /* will hold returned
error count */
HIGEAR  hIGear;     /* handle of an image */
AT_MODE nAType;
CHAR   FileName[_MAX_PATH];


 …
/* initialize of multi-page image and assign it with external
file */

nErrCount = IG_mpi_info_get( hMIGear, IG_MP_ASSOCIATION_TYPE ,
&nAType, sizeof(nAType) );
if( nAType== IG_MP_ASSOCIATE_FILE )
{
   IG_mpi_info_get( hMIGear, IG_MP_FILE_NAME, FileName,
sizeof(FileName) );
}
```

IG_mpi_is_valid

Group: Multi-page support

Declaration:

```
AT_BOOL ACCUAPI IG_mpi_is_valid( HMIGEAR hMIGear )
```

Arguments:

hMIGear	Handle to allocated multi-page image.

Description:

This function is used for checking whether the `hMIGear` value is a valid multi-page image handle. The multi-page image is valid if the **`IG_mpi_create()`** function is returned successfully and **`IG_mpi_delete()`** has not been called.

Bits Per Pixel: All

Return Value:

Returns TRUE if the given multi-page image is valid.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */

nErrCount = IG_mpi_create( &hMIGear, 0 );

if ( IG_mpi_is_valid(hMIGear) )
   printf("The multi-page image is valid!");
else
   printf("It's not valid multi-page image!");
IG_mpi_delete( hMIGear );
```

IG_mpi_page_count_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_page_count_get(
[IN]   HMIGEARhMIGear,
[OUT]  LPUINT lpPageCount
 )
```

Arguments:

hMIGear	Handle to the allocated multi-page image.
lpPageCount	Pointer indicting where to return the number of pages of the given multi-page image

Description:

This function returns the number of pages in a multi-page image (the size of page array).

NOTE: The HIGEAR handle of any given page is not significant in this function, other than to identify the image; this function simply counts the array size.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count  */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i,
       j = 0;
HIGEAR hIGear;    /* handle of an image */


 …
/* initialize multi-page image and assign it with external file
*/

nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
   if (!nErrCount)
   {
      nErrCount = IG_mpi_page_get( hMIGear, i, &hIGear );
      if (IG_mpi_page_is_valid(hMIGear, i))
         j++;
   }

printf("Number of valid pages is:%i", j);
```

IG_mpi_page_count_set

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_page_count_set(
[IN]HMIGEAR hMIGear,
[IN]UINT  nPageCount
 )
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
nPageCount	The new number of pages

Description:

This function sets the size of the page array of a valid multi-page image. If the size is increased, then new pages are initialized with a default value of `NULL`. If the array is reduced, then the removed pages are not deleted with the function **`IG_image_delete()`**. This function applies only to the image, but does not affect the associated file. The number of pages is not changed.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i,

 …
/* initialize of multi-page image and assign it with external
file */

nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
if (!nErrCount)
{
   i = 0;
   while( !IG_mpi_page_is_valid(hMIGear, nPageCount - i - 1 )
&& (i < nPageCount) )
       i++;
   }
   if (!nErrCount)
       nErrCount = IG_mpi_page_count_set( hMIGear, nPageCount -
i );
}
```

IG_mpi_page_delete

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_page_delete(
[IN]HMIGEAR hMIGear,
[IN]UINT nStartPage,
[IN]UINT nCount
 )
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
nStartPage	The first page to delete
nCount	The total number of pages to delete

Description:

This function deletes the `nCount` number of elements starting with the `nStartPage` index from the page array of the multi-page image. If the deleted pages are valid images, they are deleted with the function **`IG_image_delete()`**. Pages with higher indexes are shifted by removing the number and number of pages is decreased so that all pages are numbered from `0` to `nPageCount-1`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   i;
HIGEAR hIGear;         /* handle of an image */

 …
/* initialize of multi-page image and assign it with external
file */

nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );

/* delete all pages and shrink array to entries pages */
```

```
nErrCount = IG_mpi_page_delete( hMIGear, 0, nPageCount );
```

IG_mpi_page_get

Group: Multi-page support

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_mpi_page_get(
[IN]   HMIGEAR hMIGear,
[IN]   UINT   nPage,
[OUT]  LPHIGEAR lpHIPage
 )
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
nPage	The number of the page to return
lpHIPage	The far pointer indicating where to return a handle of the page.

Description:

This function returns the value of a page array with index `nPage`. This can be either `NULL` or a valid `HIGEAR` handle.

The multi-page image is not changed. This function makes a copy of the page handle.

This function does not load the page from an external file, but just returns the current page value of the multi-page image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;        /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS;   /* will hold returned
error count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
UINT   I = 0;
HIGEAR hIGear;     /* handle of an image */

 …
nErrCount = IG_mpi_page_get( hMIGear, i, &hIGear );
 …
```

IG_mpi_page_is_valid

Group: Multi-page support

Declaration:

```
AT_BOOL ACCUAPI IG_mpi_page_is_valid(
[IN]HMIGEAR hMIGear,
[IN]UINT nPage,
)
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
nPage	The index of the page to check.

Description:

This function returns information about the page with a given index. If the page with such an index exists as well as a valid HIGEAR handle, then it returns TRUE. In all other cases FALSE is returned.

Bits Per Pixel: All

Return Value:

Returns TRUE if the given page is valid.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT nErrCount;   /* will hold returned error count */
UINT   nPageCount = IGE_SUCCESS; /* number of pages that should
get from multi-page image */
UINT   i;
HIGEAR hIGear;     /* handle of an image */

 …
/* initialize of multi-page image and assign it with external
file */

nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
   if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
   nErrCount = IG_mpi_page_unload( hMIGear, i, 1 );
```

IG_mpi_page_set

Declaration:

```
AT_ERRCOUNT IG_mpi_page_set(
[IN]   HMIGEAR hMIGear,
[IN]   UINT   nPage,
[IN]   HIGEAR hIPage
 )
```

Arguments:

hMIGear	The handle to the allocated multi-page image.
nPage	The index of the page to set
hIPage	The handle of the image to set

Description:

This function assigns a new value to the page of the multi-page image. The previous value is not deleted with the **`IG_image_delete()`** function. The size of the multi-page image is not changed, so that page arrays is not expanded when `nPage` is greater than `nPageCount-1`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error
count */
UINT   nPageCount = 0; /* number of pages that should get from
multi-page image */
HIGEAR hPage1,
       hPage2;


 …
/* initialize of multi-page image and assign it with external
file */

nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );

/* Swap first page and last page */
if (!nErrCount && nPageCount > 1)
{
   nErrCount += IG_mpi_page_get( hMIGear, 0, &hPage1 );
```

```
    nErrConut += IG_mpi_page_get( hMIGear, nPageCount - 1,
&hPage2 );
    if (!nErrCount)
    {
    nErrCount += IG_mpi_page_set(hMIGear, iPage1, hPage2 );
    nErrCount += IG_mpi_page_set(hMIGear, iPage2, hPage1 );
    }
}
```

IG_page_count_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_page_count_get (
    const LPSTRlpszFileName,
    LPUINT lpPageCount
    );
```

Arguments:

lpszFileName	Name of file
lpPageCount	Far pointer to a `UINT` variable to receive page count

Description:

This function obtains the number of pages in a multi-page file.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
UINT    nPages;     /* Will receive number of pages   */
AT_ERRCOUNT nErrcount;   /* Count of returned errors  */
nErrcount = IG_page_count_get ( “picture.tif”, &nPages );
```

IG_page_count_get_FD

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_page_count_get_FD ( AT_INT fd, LONG
lOffset, LPUINT lpPageCount );
```

Arguments:

fd	File Descriptor handle
lOffset	Offset in file at which file image begins
lpPageCount	Far pointer to a `UINT` variable to receive page count

Description:

This function obtains the number of pages in a multi-page. Call this function when the file is already opened and you have its File Descriptor handle (`fd`).

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
INT     fd;         /* File Descriptor handle  */
UINT    nPages;     /* Will receive number of pages */
AT_ERRCOUNT nErrcount;  /* Count of returned errors  */
nErrcount = IG_page_count_get_FD ( fd, 0, &nPages );
```

IG_page_count_get_mem

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_page_count_get_mem (
   LPVOID lpImage,
   AT_UINT nImageSize,
   LPUINT lpPageCount
   );
```

Arguments:

lpImage	Far pointer to start of file image in memory
nImageSize	Size of image in memory.
lpPageCount	Far pointer to a `UINT` variable to receive page count

Description:

This function obtains the number of pages in a multi-page file, when the file image is in memory. See also functions **`IG_page_count_get_FD()`** and **`IG_page_count_get()`**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
char FAR * lpMemImage;  /* Pointer to in-memory file  */
AT_UINT nImageSize;      /* Size of image in memory */
UINT    nPages;      /* Will receive number of pages    */
AT_ERRCOUNT nErrcount;   /* Count of returned errors      */
nErrcount = IG_page_count_get_mem ( lpMemImage, nImageSize,
&nPages );
```

IG_palette_entry_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_entry_get (
   HIGEAR hIGear,
   LPAT_RGB lpRGBEntry,
   UINT nIndex
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRGBEntry	Far pointer to an AT_RGB struct to receive the three color values of the palette entry. (Note that this is not an AT_RGBQUAD struct. Also note that the order of the bytes is Blue, Green, Red in an AT_RGB struct.
nIndex	Which palette entry to get, 0 to 255

Description:

This function obtains from image **hIGear's** DIB palette, the palette entry indicated by **nIndex**.

Bits Per Pixel: 1, 4, 8-bit gray level, 8i

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;           /* HIGEAR handle of image    */
AT_RGB rgbPaletteColor;  /* Will hold returned color   */
AT_ERRCOUNT nErrcount;        /* Returned count of errors on
stack*/
/* Get palette entry 255: */
nErrcount = IG_palette_entry_get ( hIGear, rgbPaletteColor, 255
);
```

IG_palette_entry_set

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_entry_set (
   HIGEAR hIGear,
   const LPAT_RGB lpRGBEntry,
   UINT nIndex
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpRGBEntry	Far pointer to an AT_RGB struct containing the three color values to be set into the palette entry. Note that this is not an AT_RGBQUAD struct. Also note that the order is Blue, Green, Red in an AT_RGB struct.
nIndex	Which palette entry to set, 0 to 255

Description:

This function sets a single palette entry in image **`hIGear's`** DIB palette.

Bits Per Pixel: 1, 4, 8-bit gray level, 8i

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;              /* HIGEAR handle of image  */
AT_RGB rgbPaletteColor;     /* Will hold returned color */
AT_ERRCOUNT Errcount;         /* Returned count of errors
onstack */
/* Set palette entry 255 to a medium-bright yellow, intensity
175:  */
rgbPaletteColor.b = 0;     /* There's no blue in yellow */
rgbPaletteColor.r  =  rgbPaletteColor.g  =  175;
nErrcount = IG_palette_entry_set ( hIGear, &rgbPaletteColor, 255
);
```

IG_palette_get

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_get (
   HIGEAR hIGear,
   LPAT_RGBQUAD lpPalette
   );
```

Arguments:

hIGear	HIGEAR handle of image
lpPalette	Far pointer to the first of an array of AT_RGBQUAD structs sufficient to hold the entire palette from the image's DIB.

Description:

This function obtains the referenced image's DIB palette, storing it in your array of **AT_RGBQUAD** structs pointed to by **lpPalette**. The array must be large enough to hold the palette. For example, if an 8-bit image, **lpPalette** must point to the start of an array of 256 **AT_RGBQUAD** structs, therefore to an array of at least 256 x 4 = 1024 bytes. If the image is 24-bit, no error is set but no palette is returned.

NOTE: See also function **IG_palette_set()** function.

Bits Per Pixel: 1, 4, 8

NOTE: To obtain an image's logical palette (the palette after mapping through the display LUTs, or for a 24-bit image the palette that would be used for displaying to an 8-bit device), use function **IG_dspl_palette_create()**.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image     */
AT_ERRCOUNT nErrcount;     /* Returned count of errors on stack
*/
AT_RGBQUAD rgbqPalette[256];  /* Array of AT_RGBQUAD structs
*/
nErrcount = IG_palette_get ( hIGear, rgbqPalette );
```

IG_palette_load

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_load (
   const LPSTR lpszFileName,
   LPAT_RGBQUAD lpPalette,
   LPUINT lpNumEntries,
   BOOL   bBGR_Order,
```

```
    LPAT_MODE lpFileType
    );
```

Arguments:

lpszFileName	Name of file containing saved palette to load
lpPalette	Far pointer to array of AT_RGBQUAD structs to load into (But see the parameter bBGR_Order, below)
lpNumEntries	Far pointer to UINT variable to receive number of palette entries loaded (size of palette)
bBGR_Order	TRUE = store as AT_RGB structs (that is, Blue-Green-Red), instead of AT_RGBQUAD structs (Blue-Green-Red-Unused) This switch has no affect when reading ImageGear's text format. It loads this information as if writing to AT_RGBQUAD.
lpFileType	An IG_PALETTE_FORMAT_ constant specifying the format of the file. The constants are listed in file accucnst.h.

Description:

This function loads a palette that was saved using **IG_palette_save()** function.

Bits Per Pixel: 1, 4, 8-bit gray level, 8i

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_RGBQUAD rgbqPalette[256];     /* Will hold the palette loaded
*/
UINT   nEntries;           /* Holds number of entries in
palette*/
AT_MODE nPaletteFileType;   /* Will receive IG_PALETTE_…
constant*/
AT_ERRCOUNT1 nErrcount;           /* Returned count of errors
*/

nErrcount = IG_palette_load ( “Palfile.pal”, &rgbqPalette[0],
&nEntries, TRUE, &nPaletteFileType );
```

IG_palette_save

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_save (
```

```
const LPSTR lpszFileName,
LPAT_RGBQUAD lpPalette,
UINT   nNumEntries,
AT_MODE nFileType
);
```

Arguments:

lpszFileName	Name of file to save palette to.
lpPalette	Far pointer to array of AT_RGBQUAD structs constituting palette to save.
nNumEntries	Number of entries in palette.
nFileType	IG_PALETTE_FORMAT_ constant specifying format in which to save palette. The constants are listed in accucnst.h.

Description:

This function saves a palette to a file. Argument **nFileType** lets you select the format in which to save the palette. If you save the palette using **IG_PALETTE_FORMAT_TEXT**, its format will be as shown for the following palette of a 1-bit black-and-white image:

```
AccuSoft Palette File ver 7.0.9
0 0 0 0
1 255 255 255
```

The first line identifies the version of ImageGear that created this Palette text file. Then each succeeding line gives the entry number, followed by the Red, Green, and Blue color intensities, respectively, for that palette color.

Bits Per Pixel: 1, 4, 8

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR  hIGear;      /* Handle of image whose palette to save  */
LPAT_RGBQUAD lpPalettePntr; /* Will hold address of the DIB
palette  */
UINT   nEntries;    /* Will hold number of entries in palette  */

/* Obtain address of DIB palette and its number of entries );
IG_image_DIB_palette_pntr_get ( hIGear, &lpPalettePntr,
&nEntries );

/* Save to a file, saving only 3 bytes per entry, in order B-G-
R: */
nErrcount = IG_palette_save ( “Savedpal.pal”, lpPalettePntr,
nEntries, IG_PALETTE_FORMAT_RAW_BGR );
```

IG_palette_set

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_set (
   HIGEAR    hIGear,
   const LPAT_RGBQUAD lpPalette
   );
```

Arguments:

HIGEAR	HIGEAR handle of image
LpPalette	Far pointer to the first of an array of AT_RGBQUAD structs containing the palette you wish to load into the DIB, as the image's new DIB palette

Description:

This function loads the palette pointed to by **lpPalette** into the DIB, replacing the prior palette that was present. Your palette pointed to by **lpPalette** must be in the form of **AT_RGBQUAD** structs: 4 bytes per entry, ordered Blue-Green-Red-Unused (0). The number of consecutive **AT_RGBQUAD** structs you need is determined by the number of Bits Per Pixel in the image. For example, for an 8 bit image, you would need an array of 256 **AT_RGBQUAD** structs.

If the image is 24 bit, this function will set an error and return.

NOTE: See also function **IG_palette_set()**.

Bits Per Pixel: 1, 4, 8

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image  */
AT_ERRCOUNT nErrcount;     /* Returned count of errors on stack
*/
AT_RGBQUAD rgbqPalette[256];   /* Array of AT_RGBQUAD structs
*/
INT    pix;              /* Loop index, = pixel value  */
/* Create a gray-scale palette, and set it into image hIGear's
DIB:  */
for ( pix = 0;  pix <= 255;  pix++ )
   {
   rgbqPalette[pix].rgbBlue  =  pix;
```

```
    rgbqPalette[pix].rgbGreen =  pix;
    rgbqPalette[pix].rgbRed   =  pix;
    rgbqPalette[pix].rgbReserved  =  0;
    }
nErrcount = IG_palette_set ( hIGear, rgbqPalette );
```

IG_pixel_bits_per_channel_get

Group: Pixel Access

Declaration:

```
AT_DEPTH ACCUAPI IG_pixel_bits_per_channel_get(
HIGPIXEL hPixel);
```

Arguments:

hPixel	Handle of pixel object

Description:

This function returns the number of bits allocated for storing pixel data for a single channel in the given pixel object. It will be 8, 16, or 32.

Bits Per Pixel: All

Return Value:

Return value is the same for all channels and represents the maximal bit depth aligned to all bytes.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DEPTH bitsPerChan;   /* Number of bits per channel */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_DIB_pix_get(hImage, 0, 0, &hPix);
bitsPerChan = IG_pixel_bits_per_channel_get(hPix);
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

IG_pixel_channel_count_get

Group: Pixel Access

Declaration:

```
AT_INT ACCUAPI IG_pixel_channel_count_get(
```

```
HIGPIXEL hPixel );
```

Arguments:

hPixel	Handle of pixel object

Description:

This function returns the number of channels that are available to store values in the given pixel object.

Bits Per Pixel: All

Return Value:

Number of channels in pixel object.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DEPTH nChannels;     /* Number of channels */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_DIB_pix_get(hImage, 0, 0, &hPix);
nChannels = IG_pixel_channel_count_get(hPix);
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

IG_pixel_create

Group: Pixel Access

Declaration:

```
HIGPIXEL ACCUAPI IG_pixel_create(
AT_INT channelCount,
AT_DEPTH bitsPerChannel);
```

Arguments:

channelCount	Number of channels to allocate
bitsPerChannel	Bits per channel to allocate

Description:

This function creates a new pixel object based on specified attributes. The number of bits per channel is the amount to allocate, and it must be 8, 16, or 32.

Bits Per Pixel: All

Return Value:

Handle of new pixel object.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_INT ChannelCount;    /* Number of channels in image */
AT_INT BitsPerChannel;  /* Bits per channel in image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &ChannelCount);
nErrcount = IG_image_bits_per_channel_get(hImage,
&BitsPerChannel);
hPix = IG_pixel_create(ChannelCount, BitsPerChannel);
/* Pixel is created using same attributes as image */
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

IG_pixel_data_pointer_get

Group: Pixel Access

Declaration:

```
LPAT_VOID ACCUAPI IG_pixel_data_pointer_get(
HIGPIXEL hPixel);
```

Arguments:

hPixel	Handle of pixel object

Description:

This function returns a pointer to the pixel data stored for a pixel. The number of accessible bytes can be calculated by multiplying the number of bits allocated per channel (8, 16, or 32) by the number of channels allocated, then dividing by 8.

Bits Per Pixel: All

Return Value:

N/A

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
```

```
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_INT ChannelCount;    /* Number of channels in image */
AT_INT BitsPerChannel;  /* Bits per channel in image */
LPAT_VOID lpPixData;    /* Pixel data */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &ChannelCount);
nErrcount = IG_image_bits_per_channel_get(hImage,
&BitsPerChannel);
/* Get the first pixel of an image */
nErrcount = IG_DIB_pix_get(hImage, 0, 0, &hPix);
/* Set the bits in all of its channels to 1's */
lpPixData = IG_pixel_data_pointer_get(hPix);
memset(lpPixData, 255, ChannelCount * (BitsPerChannel / 8));
/* Write the modified pixel data back to the image */
IG_DIB_pix_set(hImage, 0, 0, hPix);
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

IG_pixel_delete

Group: Pixel Access

Declaration:

```
AT_VOID ACCUAPI IG_pixel_delete(
HIGPIXEL hPixel);
```

Arguments:

hPixel	Handle of pixel object

Description:

This function releases the allocated resources for a pixel object.

Bits Per Pixel: All

Return Value:

N/A

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
```

```
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, OUTPUT_FILENAME,
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

IG_pixel_value_get

Group: Pixel Access

Declaration:

```
AT_UINT ACCUAPI IG_pixel_value_get(
HIGPIXEL hPixel,
AT_INT channel);
```

Arguments:

hPixel	Handle of pixel object
channel	Channel index to get value from

Description:

This function returns the value of the requested channel. The range of possible values depends on the bit depth of the channel.

Bits Per Pixel: All

Return Value:

Channel value.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, OUTPUT_FILENAME,
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

IG_pixel_value_set

Group: Pixel Access

Declaration:

```
AT_VOID ACCUAPI IG_pixel_value_set(
HIGPIXEL hPixel,
AT_INT channel,
AT_UINT value);
```

Arguments:

hPixel	Handle of pixel object
channel	Channel index to set value for
value	Value to set (range depends on channel bit depth)

Description:

This function updates the value for the specified channel. The range of possible values depends on the bit depth of the channel.

Bits Per Pixel: All

Return Value:

N/A

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, OUTPUT_FILENAME,
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

IG_runs_row_get

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_runs_row_get(
HIGEAR          hIGear,
AT_PIXPOS       yPos,
*AT_RUN wRunCount,
LPAT_RUN        *lpRunEnd);
```

Arguments:

hIGear	HIGEAR handle of the image to get the run ends information from.
yPos	Y-Offset into the image
wRunCount	The number of runs in the lpRunEnd buffer
lpRunEnd	Returns a pointer to the run-end line specified by yPos.

Description:

This function returns a pointer to the run-end line specified by the yPos parameter. (Run-end encoding is used on 1-bit images only.) wRunCount returns the number of runs that lpRunEnd points to. The read-only data in the pointer returned should not be changed because it may corrupt the image.

NOTE: To safely change the data, use **IG_runs_row_set()**. Developers should be cautious when using the IG_runs_row_set() function because it is possible to corrupt the image by supplying invalid run-end data.

This function will set an error, if the image specified by hIGear is not in run-ends format or if the yPos parameter is greater than the height of the image.

The format of the run-end encoded data is as follows: Each line in the image starts with a value of type AT_RUN which stores the number of AT_RUN values used to hold the line. This value is equal to the number of runs in the line plus one (for the size value). The rest of the line consists of run ends of type AT_RUN. A run end specifies the first pixel position beyond the run of color. The run ends alternate between white and black, and start with white. The line ends with at least three run ends containing a value equal to the image's width. A 2500 pixel source line with black pixels in positions 0, 7, 23, and 30, would be encoded as runs: 10, 0, 1, 7, 8, 23, 31, 2500, 2500, 2500 (with 10 being the number of AT_RUN values used to store the encoded line.)

This function returns a pointer to the second AT_RUN value of the line in the lpRunEnd parameter, and returns the number of runs in wRunCount. The **IG_runs_row_set()** function copies the data in the lpRunEnd parameter to the second AT_RUN value of the line. The first AT_RUN value is set to the wRunCount parameter.

Bits Per Pixel: 1

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
AT_PIXPOS y;
WORD      wRunCount;
LPWORD    lpRunEnd, lpRunBuffer;
AT_DIMENSION nWidth, nHeight;
IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);
/* calculate the maximum size of a raster line */
lpRunBuffer = (LPWORD)malloc((nWidth + 3) * sizeof(WORD));
/* invert the image */
for (y = 0; y < nHeight; y++)
{
   IG_runs_row_get(hIGear, y, &wRunCount, &lpRunEnd);
   if (lpRunEnd[0] != 0)
   {
      lpRunBuffer[0] = 0;
      for (wRunCount = 0; lpRunEnd[wRunCount] !=
         nWidth; wRunCount++)
         lpRunBuffer[wRunCount + 1] = lpRunEnd[wRunCount];
      wRunCount++;
      lpRunBuffer[wRunCount++] = (WORD)nWidth;
      lpRunBuffer[wRunCount++] = (WORD)nWidth;
      lpRunBuffer[wRunCount++] = (WORD)nWidth;
   }
   else
   {
      memcpy(lpRunBuffer, lpRunEnd + 1, (wRunCount - 1) *
sizeof(WORD));
      lpRunBuffer[wRunCount - 1] = nWidth;
   }
   IG_runs_row_set(hIGear, y, wRunCount, lpRunBuffer);
}
InvalidateRect(hWnd, NULL, FALSE);
free(lpRunBuffer);
```

IG_runs_row_set

Group: Pixel Access

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_runs_row_set(
    HIGEAR hIGear,
    AT_PIXPOS yPos,
    AT_RUN wRunCount,
    LPCAT_RUN lpRunEnd
    );
```

Arguments:

hIGear	HIGEAR handle of the image containing the raster line you would like to modify.
yPos	Y-Offset into the image
wRunCount	The number of runs in the lpRunEnd buffer.
lpRunEnd	The Run- end pointer to copy into the image.

Description:

This function sets the run-end data of the line specified by the yPos parameter. The wRunCount parameter should be set to the number of runs contained in the lpRunEnd buffer. Run-end encoding is used on 1-bit images only. This function is faster than using **IG_DIB_raster_set()** for a 1-bit image. This function copies the data in the lpRunEnd parameter to the second AT_RUN value of the line. The first AT_RUN value is set to the wRunCount parameter.

NOTE: Developers should be cautious when using this function because it is possible to corrupt the image by supplying invalid run-end data.

Both of these functions will set an error if the image specified by hIGear is not in run-ends format or if the yPos parameter is greater than the height of the image.

The format of the run-end encoded data is as follows: Each line in the image starts with a value of type AT_RUN which stores the number of AT_RUN values used to hold the line. This value is equal to the number of runs in the line plus one (for the size value). The rest of the line consists of run ends of type AT_RUN. A run end specifies the first pixel position beyond the run of color. The run ends alternate between white and black, and start with white. The line ends with at least three run ends containing a value equal to the image's width. A 2500 pixel source line with black pixels in positions 0, 7, 23, and 30, would be encoded as runs: 10, 0, 1, 7, 8, 23, 31, 2500, 2500, 2500 (with 10 being the number of AT_RUN values used to store the encoded line.)

NOTE: See also **IG_runs_row_get()** function.

Bits Per Pixel: 1

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

See the example under **IG_runs_row_get()**.

IG_save_FD

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_FD (
    HIGEAR hIGear,
    AT_INT fd,
    UINT nPageNum,
    UINT nReserved,
    AT_LMODE lFormatType
    );
```

Arguments:

hIGear	HIGEAR handle of image to save
fd	File Descriptor handle of (already opened) file to save to
nPageNum	If saving to a multi-page file, this tells the page number to insert this page as. (Remember that the first page in a multi-page file is page 1, not 0.)
nReserved	Reserved, should always be set = 0 for now.
lFormatType	A constant such as IG_SAVE_TIF_UNCOMP or IG_SAVE_JPG, telling format to save in, and compression scheme if applicable. All AT_LMODE IG_SAVE_... constants are defined in accucnst.h.

Description:

This function saves the image rectangle of **HIGEAR** image **hIGear** to a file that has already been opened, and for which your application has the File Descriptor handle. (This is an **INT** variable, which you provide as argument fd.)

When saving to an existing file having a multi-page format, this function permits you to insert your image into the file at the page number you designate by argument **nPageNum**. If you want to append your image as the final page of the file,

set **nPageNum** = **IG_APPEND_PAGE**. **lFormatType** should specify the format type and compression of the already existing file. If you do not know the format type you can first make a call to **IG_info_get_FD_ex()**.

NOTE: Please see a pseudo-code example of saving a multi-page files in the end of the section Saving to a Disk File Using a File Descriptor Handle.

When saving to a non-multi-page format, this function will save a new single-image file of the format type and compression specified by **lFormatType**. Any previous version of the file will be lost. When saving to a non-multi-page format, set **nPageNum** = 1.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

NOTE: In order for an ImageGear append page operation to work properly, it is absolutely necessary that your File Descriptor handle is pointing to the very beginning of the first image. If the first image has a header, the FD should point to the beginning of the header. If there is no header, the FD should point to the beginning of the bitmap data. The FD should never point to “wrapper data”.

Appending and Inserting: While IG_APPEND_PAGE assures you that your loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to nPageNum will cause an append: if you set nPageNum to less than 1, or if you set nPageNum to greater than the number of pages in the file that you are saving to.

To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set nPageNum to a value between 1 and the number of the last page in the file.

NOTE: ImageGear does not support the *writing* of tiled images.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

See the examples in the section Saving to a Disk File Using a File Descriptor Handle.

IG_save_FD_CB_ex

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_FD_CB_ex(
AT_INT fd,
UINT nPage,
UINT nReserved,
AT_LMODE lFormatType,
LPFNIG_RASTER_GET lpfnRasterGet,
LPFNIG_DIB_GET_EX lpfnDIBGetEx,
LPVOID lpPrivateData
   );
```

Arguments:

fd	File Descriptor handle for an already opened file
nPage	Page number to save as, if a multi-page file; otherwise set to 1.
nReserved	Reserved, should be set to 0 for now.
lFormatType	Type and compression to save as, such as IG_SAVE_BMP_RLE. For the full list see file accucnst.h or the Table Format & Compression Type Constants (Image Saving Functions).
lpfnRasterGet	Pointer to a function of type LPFNIG_RASTER_GET, which will be called for each raster line of the imag, before that line is saved.
lpfnDIBGetEx	Pointer to a function of type LPFNIG_DIB_GET_EX, which will be called just prior to saving the DIB header.
lpPrivateData	Pointer to a private data area. This pointer will be passed to the above two callback functions each time they are called.

Description:

This function operates similarly to **`IG_save_FD()`**, except that the data constituting the file is provided to ImageGear by your callback functions.

First, your `lpfnDIBGet()` callback is called. This function supplies ImageGear with the image's width, height, bits per pixel, and all DIB information in the form of a DIB info object. If the image requires a palette, this callback function also supplies the palette.

ImageGear then writes a header out to file `fd`, in the `lFormatType` format. Next, `lpfnRasterGet()` is called once for each raster line. ImageGear gets the raster line from the callback function. Then, it compresses the raster line (according to `lFormatType`) and writes the line to `fd`. (Note that the calls for the raster lines are not necessarily in order.)

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is `IGE_SUCCESS`.

NOTE: In order for an ImageGear append page operation to work properly, it is absolutely necessary that your File Descriptor handle is pointing to the very beginning of the first image. If the first image has a header, the FD should point to the beginning of the header. If there is no header, the FD should point to the beginning of the bitmap data. The FD should never point to "wrapper data".
Appending and Inserting: While `IG_APPEND_PAGE` assures you that your loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to `nPageNum` will cause an append: if you set `nPageNum` to less than 1, or if you set `nPageNum` to greater than the number of pages in the file that you are saving to.
To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set `nPageNum` to a value between 1 and the number of the last page in the file.

NOTE: ImageGear does not support the writing of tiled images.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HFILE fd;               /* File descriptor */
DWORD dwPrivate[10];    /* Some private data */
fd = _lcreat("test.jpg", 0);
if (fd != HFILE_ERROR)
{
    nErrcount = IG_save_FD_CB_ex(
        fd,               /* File descriptor             */
        1,                /* Page number to save         */
        1,                /* Reserved. Always set to 1   */
        IG_SAVE_BMP_UNCOMP, /* File format to save as    */
        MyRasterGet,      /* Called for each raster line */
        MyDIBGetEx,       /* Called when header is saved */
        dwPrivate);       /* Callback data               */
    _lclose(fd);
}

AT_ERRCOUNT ACCUAPI MyDIBGetEx(
    LPVOID           lpPrivate, /* Private data passed in    */
    HIGDIBINFO*      lphDIB     /* DIB info object to return */
    )
{
```

```
    /* Copy handle of an existing DIB info object to lphDIB  */
    return 0;
}

AT_ERRCOUNT ACCUAPI MyRasterGet(
    LPVOID           lpPrivate, /* Private data passed in    */
    LPAT_PIXEL       lpRaster,  /* Raster line to set        */
    AT_PIXPOS        row,       /* Y position in the image   */
    DWORD            rasterSize /* Size of the raster line   */
    )
{
    /* Copy pixel data to write for this raster to lpRaster */
    return 0;
}
```

IG_save_file

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_file ( HIGEAR hIGear, LPSTR
lpszFilename, AT_LMODE lFormatType );
```

Arguments:

hIGear	HIGEAR handle of the image to save
lpszFilename	Far pointer to the filename (you may include path with filename) in which to save.
lFormatType	A constant such as IG_SAVE_TIF_UNCOMP or IG_SAVE_BMP_RLE, telling format to save in, and also the compression scheme if applicable. See file accucnst.h for a list of all IG_SAVE_... constants.

Description:

This function will store the image rectangle of the image referenced by **hIGear** to disk using the name **lpszFilename**. **lFormatType** is used to set the format and compression (if applicable) of the output file. If you want to have ImageGear use the file extension provided in your filename string (**lpszFilename**) to determine the file format to save to, set **lFormatType** = **IG_SAVE_UNKNOWN**.

When an image is saved to a multi-page file format (for example, TIFF or DCX), if the file already exists then the new image is appended as a new page in the file. When an image is saved to a non-multi-page file format, if the file already exists it is simply overwritten; the previous version of the file is lost.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

NOTE: Some file formats, such as TXT, JPEG, and others, may be saved with additional control, using **IG_fltr_ctrl_get()** and **IG_fltr_ctrl_set()**. See the description also in the section Using Format Filters API for Filter Control.

Example:

```
HIGEAR hIGear;      /* Handle of image to save */
AT_ERRCOUNT nErrcount;    /* Count of errs on stack upon ret
from func */
/* Save image to file “picture.bmp” in BMP format without
compression: */
nErrcount = IG_save_file ( hIGear, “picture.bmp”,
IG_SAVE_BMP_UNCOMP );
```

IG_save_file_size_calc

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_file_size_calc (
   HIGEAR hIGear,
   AT_LMODE lFormatType,
   LPAT_UINT   file_size
   );
```

Arguments:

hIGear	HIGEAR handle of the image on which to calculate the size.
lFormatType	A constant such as IG_SAVE_BMP_RLE or IG_SAVE_JPG, determining the file format and compression (if applicable) that will be used. The IG_SAVE_... constants associated with file formats support ed by ImageGear are listed in file accucnst.h.
file_size	Returned argument showing the maximum possible size of the file.

Description:

This function is used to determine the maximum size of the file that would result if the call **IG_save_file(hIGear,"filename.ext",lFormatType)** were executed just now. This call may be used prior to calling **IG_save_mem()** to determine the amount of memory that needs to be allocated.

Bits Per Pixel: 1, 4, 8, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

See the example for function **IG_save_mem()**.

IG_save_JPEG_quality_get

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_JPEG_quality_get ( LPUINT lpQuality
);
```

Arguments:

lpQuality	A long pointer which returns the current setting for JPEG quality.

Description:

This function returns the current setting for JPEG quality. This refers to the amount of data loss that will occur during JPEG compression. The original type of JPEG compression, which is what ImageGear uses, is a lossy scheme. This means that some data will always be lost during compression. Please see the section JPEG Quality.

NOTE: Use **IG_save_JPEG_quality_set()** to change the current setting.

NOTE: JPEG Quality is only used when an image is being saved with JPEG compression, not when an image has been compressed.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
UINT lpQuality;
IG_save_JPEG_quality_get(&lpQuality);
```

IG_save_JPEG_quality_set

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_JPEG_quality_set ( UINT nQuality );
```

Arguments:

nQuality	An integer value from 1 to 100, where 100 represents the “highest quality” or least amount of pixel data lost.

Description:

This function sets the quality level for JPEG compression. This refers to the amount of data loss that will occur during JPEG compression. The original algorithm for JPEG compression, which is what ImageGear uses, is a lossy scheme. This means that some data will always be lost during compression. Use this function to set the level of loss, where 100 means the least amount possible of pixel data will be lost during compression, and 1 allows the most loss, (resulting in the smallest possible file after compression). Please see the section JPEG Quality.

NOTE: JPEG Quality is only used when an image is being saved with JPEG compression, not when an image has been compressed.

JPEG quality may also be set using **IG_fltr_ctrl_set()** with the string "QUALITY" as the second argument:

NOTE: A setting of 100 does not give you "lossless" JPEG. The JPEG compression scheme, by design, always results in some loss. If you wish to save as lossless, use IG_fltr_ctrl_set () to set "TYPE" control parameter to IG_JPG_LOSSLESS before saving.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
UINT nQuality;
IG_save_JPEG_quality_set(95);
```

IG_save_mem

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem (
   HIGEAR hIGear,
   LPVOID lpImage,
   AT_UINT  nImageSize,
   AT_UINT  nBufferSize,
   UINT   nPageNum,
   UINT   nReserved,
   AT_LMODE lFormatType,
   LPAT_UINT lpActualSize
   );
```

Arguments:

hIGear	HIGEAR handle of the image to save to memory
lpImage	Far pointer to first byte of memory area in which to save
nImageSize	Size of image (if exists).
nBufferSize	Size of memory block.
nPageNum	If saving to a multi-page file, set this to the page number to insert this page as. (Remember that the first page in a multi- page file is page 1, not 0.) Otherwise set = 1.
nReserved	Reserved, should always be set = 0 for now.
lFormatType	A constant such as IG_SAVE_TIF_UNCOMP or IG_SAVE_JPG, indicate the format in which to save, and compression scheme if applicable. See file accucnst.h for a list of all IG_SAVE_... constants.
lpActualSize	Size of new file in memory

Description:

This function is very similar to **IG_save_FD()**, but the image file that results from the call will be saved in memory beginning at location ***lpImage**, rather than on a mass storage device. You provide the total size of your memory area, **nBufferSize**, so ImageGear can avoid writing beyond the area you have reserved for the file image. The image file that results will be identical to what would have been written to disk had you used **IG_save_FD()**: it will begin with a header, and will be in the format you have declared by argument **lFormatType**. After writing the entire new file to memory, the actual size of this in-memory file is returned to you in the **UINT** variable pointed to by **lpActualSize**.

If the file format is multi-page, and if there already is a valid file of that format at location ***lpImage**, then the **HIGEAR** image you are saving will be inserted as the page number you've indicated by **nPageNum**. If you want to append your image to the multi-page file, set **nPageNum** = **IG_APPEND_PAGE**.

If the file format is not multi-page, then any file image already at location **lpImage** will be overwritten. Set **nPageNum** = 1 for non-multi-page file formats.

It is your application's responsibility to allocate the memory to hold the file image, and to free this memory when it is no longer needed. You may call function **IG_save_file_size_calc()** to determine the maximum amount of memory you need to allocate. (If you have not allocated enough memory an error will be set and ***lpImage** will contain an unfinished image. The image left in memory after this condition should not be used.)

NOTE: In order for an ImageGear append page operation to work properly, it is absolutely necessary that your File Descriptor handle is pointing to the very beginning of the first image. If the first image has a header, the FD should point to the beginning of the header. If there is no header, the FD should point to the beginning of the bitmap data. The FD should never point to "wrapper data".

Appending and Inserting: While IG_APPEND_PAGE assures you that your

loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to nPageNum will cause an append: if you set nPageNum to less than 1, or if you set nPageNum to greater than the number of pages in the file that you are saving to.

To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set nPageNum to a value between 1 and the number of the last page in the file.

NOTE: ImageGear does not support the *writing* of tiled images.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear;          /* ImageGear handle  */
AT_ERRCOUNT nErrcount;      /* # of errors on stack */
AT_LMODE lFormatType;    /* format to save to */
AT_UINT nMaxSize,       /* Max file size might need */
       nActualSize;   /* Size of image */
HGLOBAL hBuf;            /*  handle of mem area */
AT_UINT nBuf;            /* Size of memory block */
lFormatType = IG_SAVE_BMP_RLE;/* Will save in BMP format, with
RLE compression:  */
/* Calculate max #bytes file might need: */
nErrcount = IG_save_file_size_calc ( hIGear,lFormatType,
&nMaxSize);
if (nErrcount == 0)     /* If no errs: */
{
/* Allocate sufficient memory:  */
hBuf = GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT, dwMaxSize);
/* The mem is movable. Lock it, going to reference it:    */
lpBuf = GlobalLock(hBuf);      /* lpBuf = present location  */
/* Now save the image to the mem area:  */
nErrcount = IG_save_mem ( hIGear, lpBuf, 0, nMaxSize, 1, 0,
lFormatType, &lpActualSize );
if (nErrcount == 0);    /* if successful: */
{
/* Resize the buffer to actual #bytes used:  */
   hBuf = GlobalReAlloc ( hBuf, lpActualSize, GMEM_MOVEABLE);
}
GlobalUnlock(hBuf);     /* Unlock buffer so moveable  */
}
If (nErrcount) { …}      /* Process any errors   */
```

IG_save_mem_CB_direct

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem_CB_direct(
LPVOID lpImage,
AT_UINT nImageSize,
AT_UINT nBufferSize,
UINT nPage,
UINT nReserved,
AT_LMODE lFormatType,
LPFNIG_DIRECT_RASTER_GET lpfnRasterGet,
LPFNIG_DIB_GET    lpfnDIBGet,
LPVOID lpPrivateData,
LPAT_UINT lpActualSize );
```

Arguments:

lpImage	Far pointer to first byte of memory address to save image to
nImageSize	Size of image (if exists).
nBufferSize	Size of memory block.
nPageNum	Page number to save as, if a multi- page file; otherwise set to 1.
nReserved	Reserved, should be set to 0 for now.
lFormatType	Type/compression to save as, such as `IG_SAVE_BMP_RLE`. See file `accucnst.h` for a list of all `IG_SAVE_...` constants.
lpfnRasterGet	Far pointer to a function of type **`LPFNIG_RASTER_GET`**, which will be called for each raster line of the image, before that line is saved
lpfnDIBGet	Far pointer to a function of type **`LPFNIG_DIB_GET`**, which will be called just prior to saving the DIB header
lpPrivate	Far pointer to a private data area which is passed to the above two callback functions each time they are called
lpActualSize	Size of new file in memory.

Description:

The operation of this function is similar to that of **`IG_save_FD_CB_ex()`**, except that the saving is to the memory area indicated by pointer **`lpImage`**, instead of to the file **`fd`**. After writing the entire new file to memory, the actual size of this in-memory file is returned to you in the **`DWORD`** variable pointed to by **`lpActualSize`**.

See the description for function **`IG_save_FD_CB_ex()`**, and also the description for function **`IG_save_mem()`**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24, 32

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

IG_save_mem_CB_ex

Group: Format Filters

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem_CB_ex(
LPVOID              lpImage,
AT_UINT             nImageSize,
AT_UINT             nBufferSize,
UINT                nPage,
UINT                nReserved,
AT_LMODE            lFormatType,
LPFNIG_RASTER_GET lpfnRasterGet,
LPFNIG_DIB_GET_EX lpfnDIBGetEx,
LPVOID              lpPrivateData,
LPAT_UINT           lpActualSize
);
```

Arguments:

lpImage	Address to save image
nImageSize	Size of image (if exists)
nBufferSize	Size of memory block
nPage	Page number to save as, if a multi-page file; otherwise set to 1
nReserved	Reserved, should be set to 0 for now
lFormatType	Type/compression to save as, such as IG_SAVE_BMP_RLE. See file accucnst.h or the Table Format & Compression Type Constants (Image Saving Functions) for a list of all IG_SAVE_... constants.
lpfnRasterGet	Pointer to a function of type LPFNIG_RASTER_GET, which will be called for each raster line of the image, before that line is saved
lpfnDIBGetEx	Pointer to a function of type LPFNIG_DIB_GET_EX, which will be called just prior to saving the DIB header
lpPrivateData	Pointer to a private data area which is passed to the above two callback functions each time they are called
lpActualSize	Size of new file in memory

Description:

The operation of this function is similar to that of `IG_save_FD_CB_ex()`, except that the saving is to the memory area indicated by pointer lpImage, instead of to the file fd. After writing the entire new file to memory, the actual size of this in-memory file is returned to you in the DWORD variable pointed to by `lpActualSize`.

See the description for function `IG_save_FD_CB_ex()`, and also the description for function `IG_save_mem()`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

See the example for IG_save_mem(), and add the following statements:

```
LPFNIG_DIB_GET_EX MyDIBGet;    /* Will provide DIB header */
LPFNIG_RASTER_GET MyRasterGet; /* Will provide each raster */
DWORD dwPrivate[10];           /* Some private data */
nErrcount = IG_save_mem_CB(lpBuf, 0, nMaxSize, 1, 0,
lFormatType, MyRasterGet, MyDIBGet, (LPVOID) &dwPrivate[0],
lpActualSize);
```

IG_save_tag_CB_register

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_tag_CB_register (
    LPFNIG_TAG_GET lpfnTagGet,
    LPFNIG_TAG_USER_GET lpfnTagUserGet,
    LPVOID    lpPrivate
    );
```

Arguments:

lpfnTagGet	Far pointer to callback function to be called during save operationS, prior to saving each Tag
lpfnTagUserGet	Far pointer to callback function to be called to obtain additional user Tags
lpPrivate	Far pointer to private data (passed to callback function)

Description:

This function registers a callback function to be called for each Tag while saving a file. A default value of the Tag is supplied, and can be changed in the callback function. See the description for callback function type **LPFNIG_TAG_GET**.

The second callback function you supply is of type **LPFNIG_TAG_USER_GET**, and permits you to provide additional tags. This can be used to add additional TIFF Tags when saving in TIFF format.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
LPFNIG_TAG_GET MyTagChanger;        /* To change existing tags
*/
LPFNIG_TAG_USER_GET  MyTiffTagger;       /* Can add new tags */
{
char far *    lpPrivate;             /* Pointer to private area
*/
 …
IG_save_tag_CB_register ( MyTagChanger, MyTiffTagger, (LPVOID)
lpPrivate );
 …
}
```

IG_save_thumbnail_set

Group: Image Saving

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_thumbnail_set (
   BOOL   bToggle,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight
   );
```

Arguments:

bToggle	Thumbnail Flag: TRUE = enable thumbnails, FALSE = disable thumbnails
nWidth	Width of thumbnail rectangle
nHeight	Height of thumbnail rectangle

Description:

This function allows you to save a thumbnail (miniature) version of the image together with the full image, if the format that you are saving to supports thumbnails. Currently, ImageGear reads and writes two file formats that support thumbnails: TGA & JPEG. The size of the new thumbnail will be **`nWidth`** x **`nHeight`**. The values for **`nWidth`** and **`nHeight`** are unlimited in JPEG-JFIF, but a TGA thumbnail cannot exceed 64 x 64 pixels. Targa and JPEG-JFIF thumbnails are always 8-bit images. If the image data has a bit depth of greater than 8, ImageGear will automatically reduce the number of bits when creating the thumbnail.

When you load an image using one of the **`IG_load_…()`** functions, you will not automatically load any thumbnail that accompanies the main image. To load the thumbnail you must subsequently call one of the **`IG_load_thumbnail_…()`** functions. Similarly, when calling any of the **`IG_save_…()`** functions, the thumbnail will not be saved unless you ***first*** make a call to **`IG_save_thumbnail_set()`**function.

NOTE: If you are saving a file that has a thumbnail, but you don't make a call to **`IG_save_thumbnail_set()`**, only the image will be saved to the destination file; no thumbnail will be saved.

The example code shows an image named "picture.tif" being loaded first, and then its accompanying thumbnail is loaded. When it is ready to be saved (any desired changes have been made), **`IG_save_thumbnail_set()`** is called and then **`IG_save_file()`**.

Bits Per Pixel: 8, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
HIGEAR hIGear1;    /* Will hold handle ret'd by IG_load_file  */
HIGEAR hIGear2;     /* Handle returned by IG_load_thumbnail   */
AT_ERRCOUNT nErrcount;  /* Count of errs on stack upon ret from
func */
AT_DIMENSION lWid, lHi;  /* Width and height of thumbnail   */
BOOL bToggle;    /* TRUE = thumbnail; FALSE = no thumbnail */
lWid = 32;         /* set width of thumb to 32 pixels        */
lHi = 32;          /* set height of thumbnail to 32 pixels   */
bToggle = TRUE;     /* do save a new thumbnail to new file   */
nErrcount = IG_load_file ( "picture.tif" , &hIGear1 );
if (nErrcount == 0)    /* loading the file was successful   */
   { nErrcount = IG_load_thumbnail ( "picture.tif", &hIGear2 );
}
}
{…perform image processing transformations on image…}
if (nErrcount == 0)          /* image processing was successful
*/
   /* To the file to be saved, add a thumbnail version of
processed image*/
   { nErrcount = IG_save_thumbnail_set ( bToggle, lWid, lHi );
}
}
if (nErrcount == 0)    /* thumbnail set for IMAGE was successful
*/
   {nErrcount=IG_save_file ( &hIGear1, "picture.tif",
IG_FORMAT_TIF_RLE );
}
```

IG_status_bar_CB_register

Group: Callback

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_status_bar_CB_register (
   LPFNIG_STATUS_BAR lpfnStatusBar,
   LPVOID    lpPrivate
   );
```

Arguments:

lpfnStatusBar	Far pointer to a function to be established as your status bar callback function. ImageGear will call this function once for each raster (pixel row) processed during load, save, print, and image processing operations. The argument list and return value of this function must be as shown in the definition of function type **LPFNIG_STATUS_BAR**.
lpPrivate	Far pointer to private data of your own choosing.

Description:

This function establishes a status bar callback function to be called by ImageGear during load, save, and print operations. ImageGear will call the named function once for each raster (row) processed, transmitting the Y position of that raster, the total number of rasters involved in the transfer, and the value of **lpPrivate** (pointer to your private data area). Your callback function can use this data to display a status bar showing percent completion, or for any other purpose.

To change to a different status bar function, or to change to a different private data area, call **IG_status_bar_CB_register()** again with your new callback function name and/or private data area address.

To disable status bar callbacks, call **IG_status_bar_CB_register()** with argument **lpfnStatusBar** = NULL.

NOTE: See also the description for function type **LPFNIG_STATUS_BAR**.

Bits Per Pixel: 1, 4, 8, 9-16-bit gray level, 24

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
LPFNIG_STATUS_BAR    MyStatusBarCallback;    /* Declare type of
function */
{
static DWORD      dwPrivateFlags;
 …
IG_status_bar_CB_register ( MyStatusBarCallback, (LPVOID)
&dwPrivateFlags );
```

IG_thread_data_ID_associate

Group: Threads

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_thread_data_ID_associate(
    DWORD dwNewId,
    AT_BOOL bLeaveThreadStorage
    );
```

Arguments:

dwNewId	ID of a new Data storage
bLeaveThreadStorage	If TRUE, old data storage is left in memory even if unused by any of the threads

Description:

ImageGear allows you to have different groups of settings for providing mechanisms for associating a thread with a particular group. By default, each thread uses the global copy. Such a mechanism allows you to have ImageGear settings (“groups” that are identified by specific IDs) that are customized for each thread. This API allows you to associate a thread with different ImageGear settings. Allocation of new settings will result in the use of that group’s default values if it doesn’t exist prior to the call. The old copy of the settings will be deleted unless it’s (a) a global copy used by other threads, or (b) `bLeaveThreadStorage` is `TRUE`.

NOTE: The global group of ART settings cannot be used in multi-threaded application. You have to use this function to associate a new local group of settings for each new thread.

NOTE: The access to the same PDF document from multiple threads is not permitted because the multiple threads cannot share Adobe PDF Library datatypes. PDF doc created/opened in the main thread can be only used from the main thread.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
AT_ERRCOUNT nErrCount = 0;
```

```
nErrCount = IG_comm_comp_attach("ART");

. . .
AfxBeginThread( IGProcThread1, NULL );
AfxBeginThread( IGProcThread2 , NULL );

UINT IGProcThread1( LPVOID lpData )
{
    AT_ERRCOUNT nErrCnt;
    nErrCnt= IG_thread_data_ID_associate( 1, TRUE );
    nErrCnt = IG_load_file( “picture1.bmp”, &hIGear);
    . . .
    return 0;
}

UINT IGProcThread2( LPVOID lpData )
{
    AT_ERRCOUNT nErrCnt;
    nErrCnt= IG_thread_data_ID_associate( 2, TRUE );
    nErrCnt = IG_load_file( “picture2.bmp”, &hIGear);
    . . .
    return 0;
}:
```

IG_thread_data_ID_get

Group: Threads

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_data_ID_get(LPDWORD
lpdwDataId);
```

Arguments:

LpdwDataId	A pointer to a variable for receiving information about group ID of settings currently associated with a thread.

Description:

This API retrieves the ID of the group of settings used by a particular thread.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
nErrCnt = IG_thread_data_ID_get( &dwLocalId );
```

IG_thread_local_data_cleanup

Group: Threads

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_local_data_cleanup(DWORD
dwClDataId);
```

Arguments:

dwClDataId	The group ID of settings to clean up.

Description:

This API destroys a group of settings with a specific ID if it's not being use by any threads.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
nErrCnt = IG_thread_local_data_cleanup( 1 );
```

IG_thread_image_lock

Group: Threads

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_image_lock(
   HIGEAR hIGear,
   AT_MODE nLockMode
   );
```

Arguments:

hIGear	HIGEAR image handle
nLockMode	The lock mode (IG_THREAD_LOCK_READ or IG_THREAD_LOCK_WRITE) indicator. Several threads can perform read lock concurrently, but only one thread is allowed to do write lock.

Description:

This API locks HIGEAR for a particular operation. Two APIs (IG_thread_image_lock(), **IG_thread_image_unlock()**) are required for those rare situations in which two or more threads—at least one of which is modifying or deleting an image—are simultaneously accessing the same HIGEAR. For example, one thread can save HIGEAR to a JPEG file (read access to a HIGEAR), while another is performing rotation in the asynchronous mode (write access to a HIGEAR). Thus, the "saving" thread would be required to call lock/unlock with the *read* mode as a parameter, and the "rotation" thread would be required to call lock/unlock with the *write* mode as a parameter.

NOTE: Please note that all auxilary HIGEARs associated with the main HIGEAR (such as Alpha channel, NRA mask, and transparency mask) need to be locked separately.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
hIGear = pDoc->hMainGear;
   nErrCnt = IG_thread_image_lock( pDoc->hMainGear,
IG_THREAD_LOCK_WRITE );

   if (nErrCnt == 0)
      {
      // locked successfully
      IG_IP_rotate_multiple_90( hIGear, IG_ROTATE_90 );

      IG_thread_image_unlock( pDoc->hMainGear,
IG_THREAD_LOCK_WRITE );
...
      }
```

IG_thread_image_unlock

Group: Threads

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_image_unlock(
    HIGEAR hIGear,
    AT_MODE nLockMode
    );
```

Arguments:

hIGear	ImageGear Image handle
nLockMode	The lock mode (IG_THREAD_LOCK_READ or IG_THREAD_LOCK_WRITE) indicator. Several threads can perform read lock concurrently, but only one thread is allowed to do write lock.

Description:

Two APIs (**IG_thread_image_lock()** and IG_thread_image_unlock()) are required for those situations in which two or more threads—at least one of which is modifying or deleting an image—are simultaneously accessing the same HIGEAR. For example, one thread can save HIGEAR to a JPEG file (read access to a HIGEAR), while another is performing rotation in the asynchronous mode (write access to a HIGEAR). Thus the “saving” thread would be required to call lock/unlock with the *read* mode as a parameter, and the “rotation” thread would be required to call lock/unlock with the *write* mode as a parameter.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
hIGear = pDoc->hMainGear;

   nErrCnt = IG_thread_image_lock( pDoc->hMainGear,
IG_THREAD_LOCK_WRITE );
   if (nErrCnt == 0)
      {
      // locked successfully
      IG_IP_rotate_multiple_90( hIGear, IG_ROTATE_90 );

      IG_thread_image_unlock( pDoc->hMainGear,
IG_THREAD_LOCK_WRITE );
...
      }
```

IG_tile_count_get

Group: Image Utility

Declaration:

```
LPSTR ACCUAPI IG_tile_count_get (
   const LPSTR lpszFileName,
   UINT   nPageNum,
   LPUINT lpTileRows,
   LPUINT lpTileCols
   );
```

Arguments:

lpszFileName	Name of the file containing the image
nPageNum	Page to get the count of tiles for
lpTileRows	Far pointer to a UINT variable to receive the number of tiles in a row (for tiled images)
lpTileCols	Far pointer to a UINT variable to receive the number of tile columns

Description:

This function can be called for images that may be tiled. The number of tiles across (number of tiles in a row) is returned to the **UINT** variable pointed to by **lpTileRows**, and the number of tiles down (number of tiles in a column) is returned to the **UINT** variable pointed to by **lpTileCols**. (The total number of tiles constituting the page is therefore (**lpTileRows * lpTileCols**).)

Bits Per Pixel: 1, 4, 8, 24

Return Value:

The return type is AT_ERRCOUNT. This returns the number of errors that occurred during the execution of this function.

Example:

```
UINT    nTileRows;      /* Will receive number of tile rows */
UINT    nTileCols;     /* Will receive number of tile cols  */
LPSTR   lpszFileName;  /* Holds name of file  */
AT_ERRCOUNT nErrcount;     /* Returned count of errors  */

/* Get number of tiles, first page of file:  */
nErrcount = IG_tile_count_get ( lpszFileName, 1, &nTileRows,
&nTileCols );
```

IG_tile_count_get_FD

Group: Image Utility

Declaration:

```
LPSTR ACCUAPI IG_tile_count_get_FD (
   AT_INT fd,
   LONG lOffset,
   UINT nPageNum,
   LPUINT lpTileRows,
   LPUINT lpTileCols
   );
```

Arguments:

fd	File Descriptor handle
lOffset	Offset in file of file image
nPageNum	Page to get count of tiles for
lpTileRows	Far pointer to a UINT variable to receive the number of tiles in a row rows, if the image is tiled
lpTileCols	Far pointer to a UINT variable to receive the number of tile columns

Description:

This function obtains the number of tiles constituting a page, for file formats that support tiled pages. Use this function when the file is open and you have its File Descriptor handle. See the description under **IG_tile_count_get()** for the meanings of arguments **lpTileRows** and **lpTileCols**.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

The return type is AT_ERRCOUNT. This returns the number of errors that occurred during the execution of this function.

Example:

```
INT     fd;            /* File Descriptor handle  */
UINT    nTileRows;     /* Will receive number of tile rows  */
UINT    nTileCols;     /* Will receive number of tile cols  */
AT_ERRCOUNT nErrcount;     /* Returned count of errors  */

/* Get number of tiles, first page of file:  */
nErrcount = IG_tile_count_get_FD ( fd, 0, 1, &nTileRows,
&nTileCols );
```

IG_tile_count_get_mem

Group: Image Utility

Declaration:

```
LPSTR ACCUAPI IG_tile_count_get_mem (
   LPVOID lpImage,
   AT_UINT nImageSize,
   UINT nPageNum,
   LPUINT lpTileRows,
   LPUINT lpTileCols
   );
```

Arguments:

lpImage	Far pointer to the start of the image in memory
nImageSize	Size of image in memory
nPageNum	Page to get the count of tiles for
lpTileRows	Far pointer to a UINT variable to receive the number of tiles in a row rows, if the image is tiled
lpTileCols	Far pointer to a UINT variable to receive the number of tile columns

Description:

This function obtains the number of tiles constituting a page, for file formats that support tiled pages. Use this function when the file image is in memory. See the description under **IG_tile_count_get()** for the meanings of arguments **lpTileRows** and **lpTileCols**.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

The return type is AT_ERRCOUNT. This returns the number of errors that occurred during the execution of this function.

Example:

```
char FAR * lpMemImage;       /* Pointer to file image in memory
*/
AT_UINT nImageSize;           /* Size of image in memory */
UINT    nTileRows;        /* Will receive number of tile rows */
UINT    nTileCols;       /* Will receive number of tile cols  */

AT_ERRCOUNT nErrcount;      /* Returned count of errors
*/
/* Get number of tiles, first page of file:*/
```

```
nErrcount = IG_tile_count_get_mem ( lpMemImage, nImageSize, 1,
&nTileRows, &nTileCols );
```

IG_util_colorspace_alpha_count_get

Group: Library Utility

Declaration:

```
AT_UINT ACCUAPI IG_util_colorspace_alpha_count_get(
enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	Color space ID

Description:

This function returns the number of alpha channels in the specified color space. Currently it can be either 0 or 1.

Bits Per Pixel: All

Return Value:

The number of alpha channels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_UINT nAlpha;         /* Number of alpha channels */
nAlpha = IG_util_colorspace_alpha_count_get(colorSpace);
/* nAlpha is 1 */
```

IG_util_colorspace_color_count_get

Group: Library Utility

Declaration:

```
AT_UINT ACCUAPI IG_util_colorspace_color_count_get(
enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	Color space ID

Description:

This function returns the number of color channels in the specified color space. Color channels are all image channels except alpha and extra channels.

Bits Per Pixel: All

Return Value:

The number of color channels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_UINT nColor;         /* Number of color channels */
nColor = IG_util_colorspace_color_count_get(colorSpace);
/* nColor is 3 */
```

IG_util_colorspace_contains_alpha

Group: Library Utility

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_contains_alpha(
enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	Color space ID

Description:

This function returns whether or not the color space contains an alpha channel (either premultiplied or not premultiplied).

Bits Per Pixel: All

Return Value:

Returns TRUE if the colorspace contains an alpha channel, FALSE otherwise.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
```

```
AT_BOOL bAlpha;         /* Does the color space have alpha? */
bAlpha = IG_util_colorspace_contains_alpha(colorSpace);
/* bAlpha is TRUE */
```

IG_util_colorspace_contains_extra

Group: Library Utility

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_contains_extra(
enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	Color space ID

Description:

This function returns whether or not the color space contains extra channels.

Bits Per Pixel: All

Return Value:

Returns TRUE if the colorspace contains extra channels, FALSE otherwise.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBAEx;
AT_BOOL bExtra; /* Does the color space have extra channels? */
bExtra = IG_util_colorspace_contains_extra(colorSpace);
/* bExtra is TRUE */
```

IG_util_colorspace_extra_count_get

Group: Library Utility

Declaration:

```
AT_UINT ACCUAPI IG_util_colorspace_extra_count_get(
enumIGColorSpaceIDs colorSpace,
AT_UINT totalChannelCount
);
```

Arguments:

colorSpace	Color space ID
totalChannelCount	Total number of channels in the color space

Description:

This function returns the number of extra channels in the specified color space. You must specify the total number of channels in the color space in order for this to be calculated.

Bits Per Pixel: All

Return Value:

Returns the number of extra channels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBAEx;
AT_UINT nExtra;         /* Number of extra channels */
nExtra = IG_util_colorspace_extra_count_get(colorSpace);
/* nExtra is 1 */
```

IG_util_colorspace_is_premultiplied

Group: Library Utility

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_is_premultiplied(
enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	Color space ID

Description:

This function checks whether the alpha channel in the color space is premultiplied.

Bits Per Pixel: All

Return Value:

Returns TRUE, if the alpha channel in the color space is premultiplied. Returns FALSE, if the alpha channel in the color space is not premultiplied or the color space does not contain an alpha channel.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_BOOL bPremult; /* Does color space have premult. alpha? */
bPremult = IG_util_colorspace_is_premultiplied(colorSpace);
/* bPremult is FALSE */
```

IG_util_colorspace_is_valid

Group: Library Utility

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_is_valid(
enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	Color space ID

Description:

This function checks whether ImageGear supports the color space identified by the color space constant.

Bits Per Pixel: All

Return Value:

Returns TRUE, if the color space is supported, FALSE otherwise.

Example:

```
enumIGColorSpaceIDs colorSpace;
/* Invalid color space - can't have non-premult and premult */
colorSpace = IG_COLOR_SPACE_ID_RGBA | IG_COLOR_SPACE_ID_P;
AT_BOOL bValid;         /* Is the color space valid? */
bValid = IG_util_colorspace_is_valid(colorSpace);
/* bValid is FALSE */
```

IG_util_colorspace_value_to_ids

Group: Utils

Declaration:

```
enumIGColorSpaceIDs ACCUAPI IG_util_colorspace_value_to_ids(
enumColorSpaces colorSpace
);
```

Arguments:

colorSpace	Legacy ImageGear color space ID

Description:

This function maps from legacy ImageGear color space ID (`enumColorSpaces`) to current ImageGear color space ID (`enumIGColorSpaceIDs`). It exists because the newer color space IDs are different and can store more information. See `accucnst.h` for the definitions of these enumerations.

Bits Per Pixel: N/A

Return Value:

The current ImageGear color space ID that corresponds with the given legacy ImageGear color space ID.

Example:

```
enumColorSpaces oldCS;  /* Old color space ID */
enumIGColorSpaceIDs cs; /* New color space ID */
oldCS = IG_COLOR_SPACE_CMYK;
cs = IG_util_colorspace_value_to_ids(oldCS);
/* cs is now IG_COLOR_SPACE_ID_CMYK */
oldCS = IG_COLOR_SPACE_RGBA;
cs = IG_util_colorspace_value_to_ids(oldCS);
/* cs is now IG_COLOR_SPACE_ID_RGBA */
```

IG_util_version_get

Group: Utility

Declaration:

```
VOID ACCUAPI IG_util_version_get(
[OUT]LPCHAR lpStr
);
```

Arguments:

lpStr	Pointer to an array of chars where necessary string is returned.

Description:

This utility return text information about the ImageGear version.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
CHAR   szBuf[256];
 …
IG_util_version_get( szBuf );
 …
```

IG_version_compile_date

Group: Library Utility

Declaration:

```
LPSTR ACCUAPI IG_version_compile_date (VOID);
```

Description:

This function can be called to obtain the date of compilation of the version of ImageGear you are using. The return value is a **FAR** pointer to a string in the form “Mmm dd yyyy”, such as “Jul 04 1996.”

Bits Per Pixel: N/A

Return Value:

Returns a **FAR** pointer to a string containing the compile date in the format given above. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
#include <string.h>
char  MyCompileDateString[12]; /* Will receive “Mmm dd yyyy” */
strcpy ( MyCompileDateString, IG_version_compile_date())
```

IG_version_is_eval

Group: Library Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_version_is_eval ( LPAT_BOOL
lpVersionIsEval, LPINT lpDaysTilWarning, LPINT lpDaysTilTimeOut
);
```

Arguments:

lpVersionIsEval	A pointer to an **AT_BOOL** variable; returns **TRUE** if you are using an evaluation version of the DLL, and **FALSE** if you are not.
lpDaysTilWarning	A pointer to an **INT** variable; returns the number of days until you begin receiving a warning message if this is an eval version.
lpDaysTilTimeOut	A pointer to an **INT** variable; returns the number of days until the DLL will stop working if this is an eval version.

Description:

This function is used in sample application programs, but is not normally used. It determines whether the version of the ImageGear DLL that you are using is an evaluation version. If it is an evaluation version, it returns the number of days until warnings begin, and the number of days until the product stops working.

Bits Per Pixel: Not applicable

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **IGE_SUCCESS**.

Example:

```
AT_BOOL bEvaluation;    /* Receives TRUE if an eval version */
INT     nDaysTilWarn;  /* Will receive days till warning */
INT     nDaysTilTO;    /* Will receive days till timeout */
IG_version_is_eval ( &bEvaluation, &nDaysTilWarn, &nDaysTilTO
);
```

IG_version_numbers

Group: Library Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_version_numbers (
   LPINT lpVerMajor,
```

```
    LPINT lpVerMinor,
    LPINT lpVerUpdate
    );
```

Arguments:

lpVerMajor	Far pointer to an `INT` variable in which will be stored the Major version number of the version of ImageGear that you are using
lpVerMinor	Far pointer to an `INT` variable in which will be stored the Minor version number of the version of ImageGear that you are using
lpVerUpdate	Far pointer to an `INT` variable in which will be stored the Update (bug fix) number, reflecting any updates you have received and installed in this version of ImageGear.

Description:

This function returns three integers telling you the version of ImageGear that your application is currently using, including the last update installed. The Major and Minor version numbers are, for example, the "**10**" and "**0**" respectively, for the first release of ImageGear 10.0 (ImageGear2000).

The above numbers appear in **`accucnst.h`**. Your application can compare the **`accucnst.h`** constants with the numbers returned here to verify that the proper version of the ImageGear DLL is being loaded and used.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is **`IGE_SUCCESS`**.

Example:

```
INT nVerMaj, nVerMin, nVerUpdate;
IG_version_numbers ( &nVerMaj, &nVerMin, &VerUpdate );
```

Core Component Callback Functions Reference

This section describes each callback supported by ImageGear Core component. They are presented in alphabetical order.

LPAFT_IG_ICC_GET_CB

Group: Callback

Declaration:

```
typedef VOID (LPACCUAPI  LPAFT_IG_ICC_GET_CB)(
   AT_VOID *lpPrivate,
   AT_BYTE *lpICCData,
   AT_INT DataLength
);
```

Arguments:

lpPrivate	Private callback data.
lpICCData	ICC profile data, allocated by ImageGear.
DataLength	Length of ICC profile data, in bytes.

Description:

This callback function is called every time when the format filter encounters an ICC profile in the loaded image. The ICC profile is provided in the standard ICC format as a byte array. ImageGear allocates the buffer for ICC profile data so that the application does not delete it. If the application needs to use the ICC profile after the callback call, it copies it to its own buffer.

Use **IG_fltr_ICC_callback_set()** to register this callback function.

Use **IG_fltr_ICC_callback_get()** function to retrieve your ICC callback settings.

Bits Per Pixel: **N/A**

Return Value: **N/A**

LPAFT_IG_METAD_ITEM_ADD_CB

Group: Callback

Declaration:

```
typedef AT_BOOL (LPACCUAPI LPAFT_IG_METAD_ITEM_ADD_CB)(
   LPVOID lpPrivate,
   AT_MODE FormatID,
   LPCHAR ItemName,
   DWORD ItemID,
   AT_MODE ItemType,
   LPVOID ItemValue,
```

```
AT_MODE ValueType,
DWORD ValueLength,
AT_BOOL ReadOnlyValue,
);
```

Arguments:

lpPrivate	Private callback data.
FormatID	The ID of format filter that send or get the item (`IG_FORMAT_...` constant).
ItemName	Provides the name of new item.
ItemID	Numerical ID of item.
ItemType	Specifies the type of item and reflect the status of given record. Possible values are: ■ `IG_METAD_VALUE_ITEM` – this value specifies that current item is value of simplest type and field `Value` contains actual value of item and `ValueType` contains identifier of type of this item. `ReadOnly` can be either `TRUE` (readonly) or `FALSE` (read/ write). `Name` and/or `Id` contains textual and numerical identification of item. ■ `IG_METAD_LEVEL_START` – this value specifies that current item opens sublevel of items and all next items up to correspondent item with `LEVEL_END` value belongs to this sublevel. ■ `IG_METAD_LEVEL_END` – this value close current sublevel and tells that next item belongs to higher level.
ItemValue	Value of new item. If `ItemType = IG_METAD_VALUE_ITEM` then `ItemValue` contains actual value of item of type specified by `ValueType` parameter.The `ItemValue` is stored as array of elements where each element contains values of type `ValueType`. Length of array is provided in parameter `ValueLength`.
ValueType	If `ItemType = IG_METAD_VALUE_ITEM` then contains actual type of value stored in ItemValue pointer. See Non-Image data format section for exact list of possible types.
ValueLength	Length of array in `ItemValue`.
ReadOnlyValue	If this argument is `TRUE` then actual value of item can not be changed by callback function and value is passed for information purpose only. If `FALSE` then value of item can be changed and application can provide new value through next three parameters.

Description:

This callback function is used to insert new items during **WRITE** filter operation. If return value is `TRUE` then new value is added and `FALSE` if value is not added. ImageGear will be calling **Add** callback function while it returns `TRUE` so callback function should return `TRUE` if providing of additional items is it not finished yet, and `FALSE` if it is finished. All parameters except of `lpPrivate` is used to get information about new value.

See also **IG_fltr_metad_callback_get()**, **IG_fltr_metad_callback_set()**, **LPAFT_IG_METAD_ITEM_GET_CB**, **LPAFT_IG_METAD_ITEM_SET_CB** functions and the section Processing of non-image data through filter callback functions.

Bits Per Pixel: ALL

Return Value:

Returns TRUE if new value is added to item data, FALSE if it is not. Returned TRUE will cause to call **Add** callback function one more time.

Example:

See example for **IG_fltr_metad_callback_set()** function.

LPAFT_IG_METAD_ITEM_GET_CB

Group: Callback

Declaration:

```
typedef VOID (LPACCUAPI LPAFT_IG_METAD_ITEM_GET_CB)(
   LPVOID lpPrivate,
   AT_MODE FormatID,
   LPCHAR ItemName,
   DWORD ItemID,
   AT_MODE ItemType,
   LPVOID ItemValue,
   AT_MODE ValueType,
   DWORD ValueLength,
   AT_BOOL ReadOnlyValue,
   );
```

Arguments:

lpPrivate	Private callback data.
FormatID	The ID of format filter that send or get the item (IG_FORMAT_... constant).
ItemName	Name of item.
ItemID	Numerical ID of item.

ItemType	Specifies the type of item and reflect the status of given record. Possible values are: ■ IG_METAD_VALUE_ITEM – this value specifies that current item is value of simplest type and field Value contains actual value of item and ValueType contains identifier of type of this item. ReadOnly can be either TRUE (readonly) or FALSE (read/ write). Name and/or Id contains textual and numerical identification of item. ■ IG_METAD_LEVEL_START – this value specifies that current item opens sublevel of items and all next items up to correspondent item with IG_METAD_LEVEL_END value belongs to this sublevel. ■ IG_METAD_LEVEL_END – this value close current sublevel and tells that next item belongs to higher level.
ItemValue	If ItemType = IG_METAD_VALUE_ITEM then ItemValue contains actual value of item of type specified by ValueType parameter.Value is stored as array of elements where each element contains values of type ValueType. Length of array is provided in parameter ValueLength.
ValueType	Type of element stored in array ItemValue.
ValueLength	Length of array of elements stored in ItemValue.
ReadOnlyValue	If this argument is TRUE then actual value of item can not be changed by callback function and value is passed for information purpose only. If FALSE then value of item can be changed and application can provide new value through next three parameters.

Description:

This callback function is used to get information about metadata items received during **READ** filter operation. ReadOnlyValue parameter is used to inform application that value of given item is for information only and can not be changed during future operations. It may allow to simplify future processing of this item at application level.

See also **IG_fltr_metad_callback_get()**, **IG_fltr_metad_callback_set()**, **LPAFT_IG_METAD_ITEM_ADD_CB**, **LPAFT_IG_METAD_ITEM_SET_CB** functions and the section Processing of non-image data through filter callback functions.

Bits Per Pixel: ALL

Return Value: **None**

Example:

See example for **IG_fltr_metad_callback_set()** function.

LPAFT_IG_METAD_ITEM_SET_CB

Group: Callback

Declaration:

```
typedef AT_BOOL (LPACCUAPI LPAFT_IG_METAD_ITEM_SET_CB)(
   LPVOID lpPrivate,
   AT_MODE FormatID,
   LPCHAR ItemName,
   DWORD  ItemID,
   AT_MODE ItemType,
   LPVOID ItemValue,
   AT_MODE ValueType,
   DWORD  ValueLength,
   AT_BOOL ReadOnlyValue,
   LPVOID *NewItemValue,
   LPAT_MODE *NewValueType,
   LPDWORD *NewValueLength
   );
```

Arguments:

lpPrivate	Private callback data associated with metadata callback function.
FormatID	The ID of format filter that send or get the item (`IG_FORMAT_...` constant).
ItemName	Text name of item.
ItemID	Numerical ID of item.
ItemType	Specifies the type of item and reflect the status of given record. Possible values are: ■ `IG_METAD_VALUE_ITEM` – this value specifies that current item is value of simplest type and field `Value` contains actual value of item and `ValueType` contains identifier of type of this item. `ReadOnly` can be either `TRUE` (readonly) or `FALSE` (read/ write). `Name` and/or `Id` contains textual and numerical identification of item. ■ `IG_METAD_LEVEL_START` – this value specifies that current item opens sublevel of items and all next items up to correspondent item with `LEVEL_END` value belongs to this sublevel. ■ `IG_METAD_LEVEL_END` – this value close current sublevel and tells that next item belongs to higher level.
ItemValue	If `ItemType = IG_METAD_VALUE_ITEM` then this argument contains actual value of item of type specified by `ValueType` parameter.Value is stored as array of elements where each element contains values of type `ValueType`. Length of array is provided in parameter `ValueLength`.

ValueType	If `ItemType = IG_METAD_VALUE_ITEM` then `ValueType` contains actual type of value stored in `ItemValue` pointer. See Non-Image data format section for the exact list of possible types.
ValueLength	If `ItemType = IG_METAD_VALUE_ITEM` then `VaueLength` contains number of elements in array `ValueItem` of type specified by `ValueType`.
ReadOnlyValue	If this argument is TRUE then actual value of item can not be changed by callback function and value is passed for information purpose only. If FALSE then value of item can be changed and application can provide new value through next three parameters.
NewItemValue	Pointer to new item value that application may request to set as replacement of data passed in `ItemValue` parameter. If it is not `NULL` then next two parameters contain the type of value and length of array of elements.
NewValueType	Specifies the type of new item value passed through `NewItemValue` parameter.
NewValueLength	Specifies the size of array passed through `NewItemValue` pointer.

Description:

If this callback function is defined in the format filter then during filter **WRITE** operation it is called every time when some data is ready to be writen. Application can change/provide new value for given item using `NewItemValue`, `NewValueType` and `NewValueLength` parameters. This new value is to be used as replacement for default data passed through `ItemValue` parameter in the case if `ReadOnlyValue` is `FALSE`. Callback function should return `TRUE` if callback function has changed value, and `FALSE` if it is not changed.

See also **`IG_fltr_metad_callback_get()`**, **`IG_fltr_metad_callback_set()`**, **`LPAFT_IG_METAD_ITEM_ADD_CB`**, **`LPAFT_IG_METAD_ITEM_GET_CB`** functions and the section Processing of non-image data through filter callback functions.

Bits Per Pixel: ALL

Return Value:

Returns `TRUE` if ImageGear should overwrite the default item data with your data, or `FALSE` if ImageGear should ignore your data.

Example:

See example for **`IG_fltr_metad_callback_set()`** function.

LPFNIG_DIB_CREATE

Group: Callback

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_DIB_CREATE) (
   LPVOID    lpPrivate,
   const LPAT_DIBlpDIB,
   const LPAT_RGBQUADlpRGB
   );
```

Arguments:

lpPrivate	Far pointer to private data area
lpDIB	Far pointer to start of an AT_DIB DIB header (that is, a BITMAPINFOHEADER) struct created by ImageGear
lpRGB	Far pointer to the first of the AT_RGBQUAD structs constituting the palette in the DIB. Will be NULL if the image is 24-bit.

Description:

This is one of the two types of callback functions supplied in calls to **IG_load_FD_CB()** and **IG_load_mem_CB()**. This callback function is called by ImageGear to provide your application the information it needs to create its own DIB header and palette.

On entry to this function **lpDIB** points to an **AT_DIB** struct which ImageGear has created upon reading the file's header. You can use information from this **AT_DIB** struct to create your own DIB header, but you should not alter the information at **lpDIB**.

Similarly, **lpRGB** points to the palette as obtained from the file. (**lpRGB** = NULL if no palette.) You can copy the palette, or create your own for the DIB you are creating.

NOTE: If you need to terminate the load, you can place an error on the stack yourself, using **IG_error_set().** See the description for that function.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Returns an error count.

Example:

```
BOOL  ACCUAPI MyDIB_Create (LPVOID lpPrivate, LPAT_DIB lpDIB,
LPAT_RGBQUAD lpRGB )
```

```
{
/* Can allocate memory, create a DIB header (AT_DIB) and
palette. Later, an LPFNIG_RASTER_SET function can create the
image bitmap. */
 …
return IG_error_check();
}
```

LPFNIG_DIB_CREATE_EX

Group: Callback

Declaration:

```
typedef AT_ERRCOUNT (LPACCUAPI LPFNIG_DIB_CREATE_EX)(
LPVOID lpPrivate,
HIGDIBINFO hDIB
   );
```

Arguments:

lpPrivate	Far pointer to private data area
hDIB	Extended DIB header

Description:

This callback function is called after the image header has been read. It passes image information, including palette, through `HIGDIBINFO` structure.

`hDIB` object is owned by ImageGear. The application shall not use it after exiting from the callback function, and shall not delete it.

Bits Per Pixel: All

Return Value:

Returns an error count.

LPFNIG_DIB_GET

Group: Callback

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_DIB_GET) (
   LPVOID lpPrivate,
   LPAT_DIB lpDIB,
   LPAT_RGBQUAD lpRGB
   );
```

Arguments:

lpPrivate	Far pointer to private data area
lpDIB	Far pointer to start of the DIB header, that is the AT_DIB (BITMAPINFOHEADER) struct that begins the DIB
lpRGB	Far pointer to first of the AT_RGBQUAD structs constituting the palette in the DIB. Will be NULL if it is a 24-bit image.

Description:

This is one of the two types of callback functions supplied in calls to **IG_save_FD_CB_ex()** and **IG_save_mem_CB_ex()**. This callback function is called by ImageGear prior to saving the DIB header and palette.

On entry to this function, **lpDIB** points to the image's DIB header (**AT_DIB** or **BITMAPINFOHEADER** struct), and **lpRGB** points to its DIB palette. This function is responsible for setting the DIB header fields (width, height, bits per pixel, compression, etc.) and for assuring the palette desired if it is not a 24-bit image.

NOTE: If you need to terminate the load, you can place an error on the stack yourself, using **IG_error_set()**. See the description for that function.

Bits Per Pixel: N/A

Return Value:

Returns an error count.

Example:

```
BOOL  ACCUAPI MyDIBGet (LPVOID lpPrivate, LPAT_DIB lpDIB,
LPAT_RGBQUAD lpPalette )
{
/* Modify the DIB header fields at *lpDIB as desired, and store
a
   palette at *lpPalette */
 …
return IG_error_check();
}
```

LPFNIG_DIB_GET_EX

Group: Callback

Declaration:

```
typedef AT_ERRCOUNT (LPACCUAPI LPFNIG_DIB_GET_EX)(
LPVOID lpPrivate,
```

```
HIGDIBINFO *lphDIB
);
```

Arguments:

lpPrivate	Far pointer to private data area
lphDIB	Extended DIB header

Description:

This callback function is called before writing the image header. The application shall create a new `HIGDIBINFO` object containing information about the image, and pass through the `lphDIB` parameter. ImageGear owns the `HIGDIBINFO` object after exiting from the callback, and eventually deletes it. If the image being saved has a palette, it should be passed with `HIGDIBINFO` as well.

Bits Per Pixel: N/A

Return Value:

Returns an error count.

LPFNIG_LOAD_DISP

Group: Callback

Declaration:

```
typedef VOID (ACCUAPI LPFNIG_LOAD_DISP) (
   LPVOID lpPrivate,
   HIGEAR hIGear
   );
```

Arguments:

lpPrivate	Far pointer to private data area you provided in your call to **`IG_load_file_display()`** function
hIGear	`HIGEAR` handle assigned to the image just loaded

Description:

This function is called by **`IG_load_file_display()`** after it has loaded the image and assigned to it a **`HIGEAR`** handle, but before it has displayed the image.

Using the **`HIGEAR`** handle supplied to you in this call, you can set the image and device rectangles, set display attributes, and perform other operations you choose, prior to returning. Upon your return, **`IG_load_file_display()`** will continue and will display your image, line by line, using the settings you have made.

Bits Per Pixel: 1, 4, 8, 24

Return Value: None

Example:

See the Example code in the section Working with ImageGear Callback Functions.

LPFNIG_MEM_ALLOC

Group: Callback

Declaration:

```
typedef LPBYTE (ACCUAPI LPFNIG_MEM_ALLOC)(AT_UINT nSize);
```

Arguments:

nSize	Number of bytes to allocate.

Description:

This callback function is registered by calling **IG_mem_CB_register()**. The register function must be called prior to any user-defined callback functions being used by the ImageGear library. Create a function of this type to give your application the flexibility of replacing ImageGear's memory allocation routine with your own.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

The user supplied callback function should return a pointer to the allocated block of memory.

Example:

```
/
************************************************************/
/* Memory Alloc callback function definition */
/************************************************************/
LPBYTE  ACCUAPI MyMemAlloc(AT_UINT nSize) /* number of bytes to
alloc*/
{
   /* Put your own memory allocation code here */
   return( buffer);
};
/*See also example for IG_mem_CB_register() */
```

NOTE: Your memory allocation function will only be used when large allocations (allocations greater than 1024) are performed.

LPFNIG_MEM_FREE

Group: Callback

Declaration:

```
typedef LPBYTE (ACCUAPI LPFNIG_MEM_FREE) ( LPBYTE lpBuffer);
```

Arguments:

lpBuffer	Far pointer to the buffer to be freed.

Description:

This callback function is registered by calling **IG_mem_CB_register()**. The register function must be called prior to any user-defined callback functions being used by the ImageGear library. Create a function of this type to give your application the flexibility of replacing ImageGear's memory free routine with your own.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Usually NULL.

Example:

```
/*************************************************************/
/* Memory Free callback function definition       */
*************************************************************/
LPBYTE  ACCUAPI MyMemFree( LPBYTE lpBuffer)
{
   /*MEMORY FREE CODE*/
   return NULL;
};
/*See also example for IG_mem_CB_register() */
```

LPFNIG_MEM_REALLOC

Group: Callback

Declaration:

```
typedef LPBYTE (ACCUAPI LPFNIG_MEM_REALLOC) (
```

```
LPBYTE lpBuffer,
AT_UINT nSize
);
```

Arguments:

lpBuffer	Far pointer to the buffer to be reallocated.
nSize	New byte count for realloc buffer.

Description:

This callback function is registered by calling **IG_mem_CB_register()**. The register function must be called prior to any user-defined callback functions being used by the ImageGear library. Create a function of this type to give your application the flexibility of replacing ImageGear's memory reallocation routine with your own.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

The user supplied callback function should return a pointer to the allocated block of memory.

Example:

```
*********************************************************/
/* Memory ReAlloc callback function definition        */
*********************************************************/
LPBYTE ACCUAPI MyMemReAlloc( LPBYTE lpbuffer, AT_UINT nSize)
{
   /* Put your memory reallocation code here */
   return( lpBuffer);
};
/*See also example for IG_mem_CB_register() */
```

LPFNIG_MPCB_UPDATE

Group: Callback

Please see the description for this function in the description of function **IG_mpi_CB_set()**.

LPFNIG_RASTER_GET

Group: Callback

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_RASTER_GET) (
   LPVOID lpPrivate,
   LPAT_PIXEL lpRast,
   AT_PIXPOS cyPos,
   DWORD  cRasterSize
   );
```

Arguments:

lpPrivate	Far pointer to private data area
lpRast	Far pointer to first byte of raster your function is providing
cyPos	The raster's Y position in the image (0 = top line of image)
dwRasterSize	Number of bytes in the raster

Description:

This is one of the two types of callback function supplied in calls to **IG_save_FD_CB_ex()** or **IG_save_mem_CB_ex()**functions. It is called by ImageGear to obtain from your application each raster to be saved. The line's position in the image is identified by **cyPos**. The lines may not be in order.

lpRast is a pointer to the start of your raster, and **dwRasterSize** is the number of bytes in the line. ImageGear will compress the line for you as it saves it, according to the compression scheme specified in your original call to save the image. **lpPrivate** points to the private data area supplied in that call.

NOTE: You should check the ImageGear error count (AT_ERRCOUNT) after each raster is read. If you need to terminate the load, you can place an error on the stack yourself, using **IG_error_set().** See the description for that function .

Bits Per Pixel: N/A

Return Value:

Return an error count.

Example:

```
AT_ERRCOUNT  ACCUAPI MyRasterGet (LPVOID lpPrivate, LPAT_PIXEL
lpRast, AT_PIXPOS lYpos, DWORD dwBytes ){
/* Provide raster row lYpos, by storing it where LPAT_PIXEL
points.
```

```
   Should be exactly dwBytes, counting padding. */
 …
return IG_error_check();
}
```

LPFNIG_RASTER_SET

Group: Callback

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_RASTER_SET) (
   LPVOID    lpPrivate,
   const LPAT_PIXEL lpRast,
   AT_PIXPOS cyPos,
   DWORD     cRasterSize
   );
```

Arguments:

lpPrivate	Far pointer to private data area
lpRast	Far pointer to first byte of raster that ImageGear is providing on this call
cyPos	The raster's Y position in the image (0 = top line of image)
dwRasterSize	Number of bytes in the raster

Description:

This is one of the two types of callback function supplied in calls to **IG_load_FD_CB()** or **IG_load_mem_CB()**functions. This function is called by ImageGear to provide to your application each raster as it is obtained from the file. The line's position in the image is identified by **cyPos**. The lines may not be in order.

lpRast is a pointer to the start of the raster, and **dwRasterSize** is the number of bytes in the line. You should not attempt to change the data at **lpRast**. In general, your application will use this data (along with its knowledge of the width of the image it is creating, any compression scheme, etc.) to create the appropriate raster for the image bitmap of the DIB your application is creating.

NOTE: See also callback type **LPFNIG_DIB_CREATE**.

NOTE: You should check the ImageGear error count (**AT_ERRCOUNT**) after each raster is read. If you need to terminate the load, you can place an error on the stack yourself, using **IG_error_set()**. See the description for that function.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Return an error count.

Example:

```
AT_ERRCOUNT ACCUAPI MyRasterSet (LPVOID lpPrivate, LPAT_PIXEL
lpRast, AT_PIXPOS lYpos, DWORD dwBytes )
{
/* Can use the above information to create a raster or rasters
in image bitmap of DIB that was created by LPFNIG_DIB_CREATE
callback.     */
 …
return IG_error_check();
}
```

LPFNIG_READ

Group: Callback

Declaration:

```
typedef LONG (ACCUAPI LPFNIG_READ) (
   LONG fd,
   LPBYTE lpBuffer,
   LONG lSize
   );
```

Arguments:

fd	File Descriptor handle
lpBuffer	Far pointer to buffer into which to read data
lSize	Number of bytes to read

Description:

This type of function is established by calling **IG_file_IO_register()** function. It will be called during file operations when a **READ** is required.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Return the number of bytes read, or -1 to indicate that an error occurred.

Example:

See the example for function **IG_file_IO_register()**.

LPFNIG_SEEK

Group: Callback

Declaration:

```
typedef AT_INT(ACCUAPI LPFNIG_SEEK) (
   AT_INT fd,
   AT_INT lOffset,
   INT nFlag
   );
```

Arguments:

fd	File Descriptor handle
lOffset	Offset to seek to
nFlag	0 = seek from start, 1 = seek from current position, 2 = seek from end

Description:

This type of function is established by calling **IG_file_IO_register()**. It will be called during file operations when a SEEK is required. This function should return the offset into the file after the seek has completed, or -1 to indicate that an error occurred.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Return offset into file at the completion of the seek, or -1 to indicate an error occurred.

Example:

```
HIGEAR        hIGear;       /* HIGEAR handle of image */
LPFNIG_READ   MyReadFunc;    /* To be called for READs */
LPFNIG_WRITE  MyWriteFunc;   /* To be called for WRITEs */
LPFNIG_SEEK   MySeekFunc;    /* To be called for SEEKs  */
{
/* Register Read, Write, and Seek callback functions:   */
IG_file_IO_register ( MyReadFunc, MyWriteFunc, MySeekFunc );
 …
IG_save_file ( hIGear, “picture.bmp”, IG_SAVE_BMP_UNCOMP );
 …
}
/* This will be called for each seek during the above Save:  */
LONG ACCUAPI  MySeekFunc ( AT_INT fd, AT_INT lOffset, INT nFlag
)
{
AT_INT nResultOffset;
```

```
 …
return  nResultOffset;
}
```

LPFNIG_SIZE_CHANGE

Group: Callback

Declaration:

```
typedef LONG (LPACCUAPI LPFNIG_SIZE_CHANGE)(
AT_INT fd,
AT_INT lSize
   );
```

Arguments:

fd	Descriptor ID, from open
lSize	New size of file in bytes

Description:

This type of function is established by calling **IG_file_IO_register()**. It will be called during file operations when a change of file size is required.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Returns the new size of the file, or -1 to indicate that an error occurred.

LPFNIG_STATUS_BAR

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_STATUS_BAR) (
   LPVOID lpPrivate,
   AT_PIXPOS cyPos,
   AT_DIMENSION dwHeight
   );
```

Arguments:

lpPrivate	Far pointer to private data area, as specified in your call to **IG_status_bar_CB_register()** function.
cyPos	Y position in the image of this raster (row). Calls for the rows are not guaranteed to be in a particular order.
dwHeight	Total number of rasters (rows) in the image

Description:

This is the type of the callback function you specify in calling `IG_status_bar_CB_register()`.

This callback function is called once for each raster (row) in the image. The calls will not necessarily be in row order. See also the description for function **IG_status_bar_CB_register()**.

Bits Per Pixel: N/A

Return Value:

Your **LPFNIG_STATUS_BAR()** callback function should return **TRUE** if ImageGear should continue the load, save, or print operation it is performing, or **FALSE** if ImageGear should terminate the operation, placing an **IGE_INTERRUPTED_BY_USER** error on the error stack.

Example:

```
BOOL ACCUAPI StatusBar(
   LPVOID            lpPrivate,    /* Private data passed in  */
   AT_PIXPOS            cyPos,     /* Y position in the image */
   AT_DIMENSION      dwHeight      /* Height of the image */
   );
  ...
    /* register the status bar callback function */
    err_count = IG_status_bar_CB_register(StatusBar, &si);
if (err_count == 0)
{
    fSBEnabled = TRUE;
    CheckMenuItem(GetMenu(
hWnd),ID_OPTIONS_PROGRESSBAR,MF_CHECKED);
}
          }
else
          {
   /* deregister SB function */
   /* deregister the status bar callback function */
   err_count = IG_status_bar_CB_register( NULL, NULL);
if (err_count == 0)
{
   fSBEnabled = FALSE;
```

```
   CheckMenuItem(GetMenu(
hWnd),ID_OPTIONS_PROGRESSBAR,MF_UNCHECKED);
}
         }
  ...
```

LPFNIG_TAG_GET

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_TAG_GET) (
   LPVOID lpPrivate,
   AT_MODEnIGTag,
   LPAT_MODElpDataType,
   LPVOID lpTagData,
   DWORD  dwSize
   );
```

Arguments:

lpPrivate	A far pointer to a private data area that can be used for anything you like. For example, you might store the HIGEAR handle of the image that is being loaded.
nIGTag	ImageGear supplies you with an IGTAG_ Tag ID constant from file Geartags.h, so that you will know which tag is currently being saved.
lpDataType	ImageGear supplies you with an IG_TAG_TYPE_ constant from file accucnst.h, specifying the data type for this tag (e.g., BYTE, LONG, FLOAT, etc.)
lpTagData	ImageGear supplies you with this pointer to a buffer, where you should store your data. If you read this field, you will find that it contains a default value. For example, the ImageGear default value for the Artist tag is "1996-97 AccuSoft Inc., All rights reserved."
dwSize	ImageGear tells you the size of tag data buffer, in bytes. This is your limit for the length of what you store to lpTagData.

Description:

This callback function is registered by calling **IG_save_tag_CB_register()**. When ImageGear is going to perform any save operation it will first check to see if you have registered any applicable callbacks. If you have registered a callback of type **LPFNIG_TAG_GET**, it will be called once for each “non-volatile” tag that is being written. ImageGear will not call your callback for what it terms “volatile” tags, so that you are unable to modify such tags. If you are writing a TIFF file and want to determine which tags you can write to (that is, which tags are non-volatile), please see the TIFF tags section of Geartags.h. This section is comprised of a list of constants for all registered TIFF tags and a 5-column key that gives you information about the read and write ability of each tag.

NOTE: See also the Note under TIFF File Format Reference, for an explanation of how to use the key.

While you may set the data of a non-volatile tag, your data must not exceed the length specified by **dwSize**. You must also use the proper data type, which you can check by reading the **lpDataType** parameter.

Your callback could contain a switch statement for each tag you would like to write to. Each case in the switch statement could check the ImageGear default setting of the tag and decide whether to change the data. Set your callback to **TRUE** if you want ImageGear to overwrite the data of a tag, **FALSE** if you want it to ignore your data. When you set the callback to **FALSE**, ImageGear will use its own default value for the current tag, but will still call your callback when it parses the next tag.

NOTE: Note also that some tags can be set only to a certain range of valid values. If your data is out of range for such tags, ImageGear will ignore your data.

If you would like to modify tag data as the image is being read in (for instance, you want it to be modified for display purposes), register a callback of type **LPFNIG_TAG_SET**.

If you would like to add your own user-defined tags to a TIFF file, register a callback of type **LPFNIG_TAG_USER_GET**. Your tags will be saved with the image when it is being saved.

NOTE: See also the discussion about Tag Callbacks in the section Working with ImageGear Callback Functions.

Bits Per Pixel: N/A

Return Value:

Returns **TRUE** if ImageGear should overwrite the default tag data with your data, or **FALSE** if ImageGear should ignore your data.

Example:

```
BOOL ACCUAPI TagGet(
   LPVOID         lpPrivate,     /* Private data passed in
*/
   AT_MODE        nIGTag,        /* Tag ID from geartags.h
*/
   LPAT_MODE      lpType,        /* Type of data lpTag points to
*/
   LPBYTE         lpTag,         /* Pointer to tag data
*/
```

```
   DWORD         dwSize        /* Size of tag data (bytes)
*/
   )
...
return TRUE; /* FALSE to terminate the save operation */
}
...
   IG_save_tag_CB_register( TagGet, TagUserGet,
(LPVOID)&dwPrivateFlags );
/* The following example may be used for writing additional
information into a GIF file: */
BOOL ACCUAPIGifCallbackTagGet(
   LPVOID    lpPrivate,
   AT_MODE   nIGTag,
   LPAT_MODE lpType,
   LPVOID    lpData,
   DWORD     dwSize
   )
{
   LPBYTE    lpRGB;
   static INTi=0;
   switch(nIGTag)
   {
      case IGTAG_GIF_SCREEN_WIDTH:
         *(LPWORD)lpData=777;
         break;
      case IGTAG_GIF_SCREEN_FLAGS:
         *(LPBYTE)lpData=0x80 | 0x07;
         break;
      case IGTAG_GIF_IMAGE_FLAGS:
         *(LPBYTE)lpData=0x80 | 0x07;
         break;
      case IGTAG_GIF_IMAGE_LEFT:
         *(LPWORD)lpData=111;
         break;
      case IGTAG_GIF_SCREEN_PALETTE:
         lpRGB=(LPBYTE)lpData;
         for(i=0; i<256; i++)
         {
            lpRGB[3*i]=(BYTE)(0);
            lpRGB[3*i+1]=(BYTE)(i);
            lpRGB[3*i+2]=(BYTE)(0);
         }
         break;
      case IGTAG_GIF_IMAGE_PALETTE:
         lpRGB=(LPBYTE)lpData;
         for(i=0; i<256; i++)
         {
            lpRGB[3*i]=(BYTE)i;
            lpRGB[3*i+1]=(BYTE)(0);
            lpRGB[3*i+2]=(BYTE)(0);
         }
```

```
            break;
        case IGTAG_GIF_EXT_NUMBER_BEFORE_IMG:
            *(LPWORD)lpData=4;
            break;
        case IGTAG_GIF_EXT_BEFORE_IMG:
            switch(i++%4)
            {
                case 0:
                    CtrlExt.bLabel=CTRL_EXT_LABLE;
                    CtrlExt.bPacked=249;
                                {
                case 0:
                    CtrlExt.bLabel=CTRL_EXT_LABEL;
                    CtrlExt.bPacked=249;
                    CtrlExt.wDelayTime=555;
                    CtrlExt.bColorIndex=111;
                    *(LPVOID FAR*)lpData=(LPVOID)&CtrlExt;
                    break;
                case 1:
                    TextExt.bLabel=TEXT_EXT_LABEL;
                    TextExt.wTextGridLeft=333;
                    TextExt.lpData=lpStr;
                    *(LPVOID FAR*)lpData=(LPVOID)&TextExt;
                    break;
                case 2:
                    CommExt.bLabel=COMM_EXT_LABEL;
                    CommExt.lpData=lpStr;
                    *(LPVOID FAR*)lpData=(LPVOID)&CommExt;
                    break;
                case 3:
                    ApplExt.bLabel=APPL_EXT_LABEL;
                    strcpy((CHAR*)ApplExt.Identifier,
                        "Accusoft");
                    strcpy((CHAR*)ApplExt.AuthentCode, "6.0");
                    ApplExt.lpData=lpStr;
                    *(LPVOID FAR*)lpData=(LPVOID)&ApplExt;
                    break;
            }
            break;
    }

    return TRUE
```

LPFNIG_TAG_SET

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_TAG_SET) (
    LPVOID lpPrivate,
```

```
AT_MODEnIGTag,
AT_MODEnDataType,
const LPVOIDlpTagData,
DWORD   dwSize
);
```

Arguments:

lpPrivate	A far pointer to a private data area that can be used for anything you like
nIGTag	ImageGear supplies you with an `IGTAG_` Tag ID constant from file `Geartags.h`, so that you will know which tag is currently being processed.
nDataType	ImageGear supplies you with an `IG_TAG_TYPE_` constant from file `accucnst.h`, specifying the data type for this tag (e.g., `BYTE`, `LONG`, `FLOAT`, etc.)
lpTagData	ImageGear supplies you with a copy of the data from the tag currently being read
dwSize	ImageGear tells you the size of tag data, in bytes

Description:

This callback function is registered by calling **`IG_load_tag_CB_register()`**function. When ImageGear is going to perform any load operation it will first check to see if you have registered any applicable callbacks. If you have registered a callback of type **`LPFNIG_TAG_SET`**, it will be called once for each tag (both volatile and non-volatile, see **`LPFNIG_TAG_GET`**) that is being parsed. ImageGear will not let you write data to any tag while loading.

If you would like to modify tag data as the image is being saved, register a callback of type **`LPFNIG_TAG_GET()`**.

If you would like to add your own user-defined tags to a TIFF file, register a callback of type **`LPFNIG_USER_TAG_GET`**. Your tags will be saved with the image when it is being saved.

NOTE: See also the discussion about Tag Callbacks in the section Working with ImageGear Callback FunctionsThe `HIGEAR` for the image is not created until after all of the tags have been read in.

NOTE: If you need to store any tag data, make a copy of the data, not the `lpTagData` pointer.

Bits Per Pixel: N/A

Return Value:

Returns **TRUE** if ImageGear should overwrite the default tag data with your data, or **FALSE** if ImageGear should ignore your data.

Example:

```
BOOL ACCUAPI TagSet(
   LPVOID        lpPrivate,     /* Private data passed in
*/
   AT_MODE       nIGTag,        /* Tag ID from geartags.h
*/
   AT_MODE       nType,         /* Type of data in lpTag
*/
   const LPVOID   lpTag,        /* Pointer to tag data
*/
   DWORD         dwSize         /* Size of tag data (bytes)
*/
   )
...
return TRUE; /* FALSE to terminate the operation */
}
...
IG_load_tag_CB_register( TagSet, (LPVOID)&dwPrivateFlags );
...
```

The following example illustrates loading a GIF image.

Example:

```
/* Example of LPFNIG_TAG_SET type function, which may be used
during loading GIF image: */
BOOL ACCUAPI GifCallbackTagSet(
   LPVOID    lpPrivate,
   AT_MODE nIGTag,
   AT_MODE nDataType,
   LPVOID    lpData,
   DWORD     dwDataLen
{
   WORD      wWidth;
   WORD      wLeft;
   WORD      wExtNumber;
   LPBYTE    lpRGB;

   LPAT_GIF_CTRL_EXT lpCtrlExt;
   LPAT_GIF_TEXT_EXT lpTextExt;
   LPAT_GIF_COMM_EXT lpCommExt;
   LPAT_GIF_APPL_EXT lpApplExt;
   LPVOID    lpExt;
/* Following code illustrates how to get GIF file information*/
/* Parameter nIGTag in this function informs about kind of data
on which points lpData.    */
```

```
/* If GIF image is loaded this parameter may be equal to
IGTAG_GIF_ constants, defined in  */
/* file \ACCUSOFT\GEAR\SOURCE\INCLUDE\geartags.h. For example,
when IGTag is equal to*/

/* IGTAG_GIF_SCREEN_ASPECT lpData must point (after conversion)
to BYTE - bAspectRatio field */
/* of GIF_SCREEN_DESC structure. Analogously for other
IGTAG_GIF_ constants.*/
/* IGTAG_GIF_SCREEN_BG_COLOR - for getting or setting screen
background color index*/
/* - lpData points to BYTE, */
/* 1.When GIF image is loaded: lpData points to GIF Extention
Block structure*/
/* 2.When GIF image is written: lpData must points to LPVOID,
which points to created by user*/
/* GIF Extention Block structure. This structure must exist all
time when image is written.   */
/* IGTAG_GIF_EXT_AFTER_IMG - Analogously as for
IGTAG_GIF_EXT_BEFORE_IMG*/
   switch(nIGTag)
   {
      case IGTAG_GIF_SCREEN_WIDTH:
         wWidth=*(LPWORD)lpData;
         break;
      case IGTAG_GIF_IMAGE_LEFT:
         wLeft=*(LPWORD)lpData;
         break;
      case IGTAG_GIF_SCREEN_PALETTE:
         lpRGB=(LPBYTE)lpData;
         break;
      case IGTAG_GIF_IMAGE_PALETTE:
         lpRGB=(LPBYTE)lpData;
         break;
      case IGTAG_GIF_EXT_NUMBER_BEFORE_IMG:
         wExtNumber=*(LPWORD)lpData;
         break;
      case IGTAG_GIF_EXT_BEFORE_IMG:
         lpExt=lpData;
         switch(*(LPDWORD)lpExt)
         {
            case CTRL_EXT_LABLE:
               lpCtrlExt=(LPAT_GIF_CTRL_EXT)lpExt;
               break;
            case TEXT_EXT_LABLE:
               lpTextExt=(LPAT_GIF_TEXT_EXT)lpExt;
               break;
            case COMM_EXT_LABLE:
               lpCommExt=(LPAT_GIF_COMM_EXT)lpExt;
               break;
            case APPL_EXT_LABLE:
               lpApplExt=(LPAT_GIF_APPL_EXT)lpExt;
```

```
                break;
            }
            break;
        case IGTAG_GIF_EXT_NUMBER_AFTER_IMG:
            wExtNumber=*(LPWORD)lpData;
            break;

        case IGTAG_GIF_EXT_AFTER_IMG:
            lpExt=lpData;
            switch(*(LPDWORD)lpExt)
            {
                case CTRL_EXT_LABLE:
                    lpCtrlExt=(LPAT_GIF_CTRL_EXT)lpExt;
                    break;
                case TEXT_EXT_LABLE:
                    lpTextExt=(LPAT_GIF_TEXT_EXT)lpExt;
                    break;
                case COMM_EXT_LABLE:
                    lpCommExt=(LPAT_GIF_COMM_EXT)lpExt;
                    break;
                case APPL_EXT_LABLE:
                    lpApplExt=(LPAT_GIF_APPL_EXT)lpExt;
                    break;
            }
            break;
    }

    return TRUE
```

LPFNIG_TAG_USER_GET

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_TAG_USER_GET) (
    LPVOID lpPrivate,
    LPAT_MODE lpUserTag,
    LPAT_MODE lpDataType,
    LPVOID32 FAR*lpTagData,
    LPDWORD lpSize
    );
```

Arguments:

lpPrivate	Far pointer to private data area, which can be used for anything you like, including the storage of the image's HIGEAR handle.
lpIGTag	Far pointer to an AT_MODE variable to receive a private tag number. Your value must be 32768 or higher.
lpDataType	Set to an IG_TAG_TYPE_ constant (these are listed in file accucnst.h) specifying the data type for this tag
lpTagData	Return a pointer to your data
lpSize	Tell ImageGear the size of your data

Description:

This function allows you to store a private TIFF tag with the image being saved. This callback function is registered by calling **IG_save_tag_CB_register()**. In this callback, you supply a tag number, tag type, and tag data. Note that the value of your private tag number must be higher than 32768.

When ImageGear is going to save a file it will check to see if you have registered any applicable callbacks. If you have registered a callback of type **LPFNIG_USER_TAG_GET()**, it will be called until you set it to **FALSE**. While your callback is still set to **TRUE**, it will add your user-defined tags one at a time to the TIFF image being saved.

lpIGTag is a far pointer to an **AT_MODE** in which you store a privately defined tag ID#. **lpDataType** is a pointer to your Tag data type, **lpTagData** is a pointer to your tag data, and **lpSize** is the length of your tag data.

This callback function should return **TRUE** when supplying a tag, in which case it will be called again. Return **FALSE** when you are done supplying tags. This callback will be called at least once if it has been registered.

Bits Per Pixel: N/A

Return Value:

Return **TRUE** if returning a tag, in which case this callback function will be called again in case you have more tags to supply. Return **FALSE** if you are not returning a tag and are done returning tags.

Example:

```
BOOL ACCUAPI TagUserGet(
   LPVOID         lpPrivate,     /* Private data passed in */
   LPAT_MODE      lpnIGTag,        /* Tag ID from geartags.h */
   LPAT_MODE      lpType,       /* Type of data lpTag points to
*/
   LPVOID  FAR*   lpTag,         /* Pointer to tag data */
   LPDWORD        lpSize         /* Size of tag data (bytes) */
   )
/* set Tag ID, data type, length, and pointer to your data */
{
```

```
   *lpSize = 0;
   if( (*lpnIGTag>=IGTAG_JPG_APPDATA) &&
(*lpnIGTag<=IGTAG_JPG_APPDATA_LAST) )
    {
if( (*lpnIGTag-IGTAG_JPG_APPDATA)==12)/* supplying only APP13
marker*/
       {
/* supply application marker data */
          *lpType = IG_TAG_TYPE_RAWBYTES;
          *lpTag = (LPVOID)(&TestData[0]);
          *lpSize = sizeof(TestData);
          returnTRUE;
       }
/* FALSE when no tag being returned */
    }
                      returnFALSE;
}
...
IG_save_tag_CB_register( TagGet, TagUserGet,
(LPVOID)&dwPrivateFlags );
...
```

LPFNIG_WRITE

Group: Callback

Declaration:

```
typedef LONG (ACCUAPI LPFNIG_WRITE) (
   LONG   fd,
   const LPBYTE lpBuffer,
   LONG   lSize
   );
```

Arguments:

fd	File Descriptor handle
lpBuffer	Far pointer to buffer to write from
lSize	Number of bytes to write

Description:

This type of function is established by calling **IG_file_IO_register()**. It will be called during file operations when a **WRITE** is required.

Bits Per Pixel: 1, 4, 8, 24

Return Value:

Return the number of bytes written, or -1 to indicate that an error occurred.

Example:

```
HIGEAR       hIGear;        /* HIGEAR handle of image */
LPFNIG_WRITE  MyWriteFunc; /* To be called for file WRITEs */
{
IG_file_IO_register ( NULL, MyWriteFunc, NULL );    /* Register
it */
 …
IG_save_file ( hIGear, “picture.bmp”, IG_SAVE_BMP_UNCOMP );
 …
}
/* This will be called for each write during the above Save:  */
LONG ACCUAPI  MyWriteFunc ( LONG fd, LPBYTE lpBuffer, LONG
lNumToWrite )
{
LONG nNumActuallyWritten;
 …
return  nNumActuallyWritten; /* Return count, or -1 for error
*/
```

ART Component API Function Reference

In this Chapter you can get the detailed information about all ImageGear ART component functions, macros and callbacks:

- ART Component Functions Reference
- ART Component Macros
- ART Component Callback Functions Reference

NOTE: You can only attach one ART component. ART cannot be attached when ARTX is already attached.

NOTE: ART component functions are accessible only when ART component is attached to Core ImageGear. Please see the section Using ImageGear Component Manager.

ART component functions are designated with the prefix `ART_...`.

ART component callback functions are designated with `LPFNIG_ART_...` and `LPFNART_...` prefixes. Please see the section ART Component Callback Functions Reference.

ART Component Functions Reference

ART Component API Classifications

There are several classifications for ART component functions. All these functions are divided on groups. The following table lists the different groups. Discussions explaining each group appears after the table.

Function Names beginning with	Group
ART_callback_register_	Callback Register
ART_control_option_	Control
ART_edit_	Clipboard
ART_edit_undo_	Undo
ART_file_	File

ART_global_block_	Global Named Block Management
ART_group_	Mark Grouping
ART_GUI_	Graphical User Interface
ART_image_	Image Utility
ART_mark_	Mark Management
ART_mark_block_	Named Block Management
ART_version_	Library Utility

Block Operations

ART component has two types of blocks:

- Global Named Blocks
- Named Blocks.

Global Named Blocks hold values that act as defaults for the next mark created. **Named Blocks** can be used to set specific attributes for the current mark.

Global Named Blocks work with “Global Named Block Management” functions; these function names begin with `ART_global_block_`. **Named Blocks** work with “Named Block Management” functions; these function names begin with `ART_mark_block_`.

See also the section ART Component Important Structures and Types.

Callback & Callback Register

Callbacks are low-level functions that provide control over the way that ART works. Each callback function must be "registered" in order to be automatically called by ART. Function with the name **`ART_callback_register()`** registers ART callback.

Functions labelled as "Callback", when registered, are called by ART during certain events:

- when the user is about to create a new mark,
- when they are going to delete a mark,
- when the mouse is clicked over a mark.

ART returns important data to you and you can send data back to ART. This exchange of information determines whether or not to proceed with an operation. See the section entitled ART Component Callback Functions Reference.

Clipboard

Clipboard function names begin with `ART_edit_` and allow you to copy and cut marks to the clipboard, and to paste marks into an image.

Control

Currently there are two functions in this category: **`ART_control_option_set()`** and **`ART_control_option_get()`**. The former function modifies default library settings. The latter function gets a value of the control option.

File

This set of functions begin with ART_file_. They control import and export of .ART files.

Image Utility

There is one function in this group, **ART_image_burn_in()**. This function causes marks to be "burned in" to an image when it is saved. There are many options for this function, including the abilitiy to burn in only a specified list of the available marks.

Mark Management

These functions begin with ART_mark_. They modify and query marks. The **ART_mark_modify()** updates the values in the mark attributes structure. This structure is defined in ARTAPI.H and is illustrated in the section ART Component Important Structures and Types .

The **ART_mark_query()** function returns the values in the attributes structure.

Undo

Undo function names begin with ART_edit_undo_. These functions record edits to marks, and undo a specified number of edits.

ART_callback_register

Group: Callback Register

Declaration:

```
AT_ERRCOUNT ART_callback_register(HIGEAR hIGear,
   AT_MODE   nCallbackID,
   LPFNIG_ART_CALLBACKFUNC lpfnCallback,
   LPVOID    lpPrivate
   );
```

Arguments:

hIGear	HIGEAR handle to the image
nCallbackID	An ART-defined constant of type AT_MODE that tells which type of callback is being registered
lpfnCallback	A far pointer to your callback function to register
lpPrivate	A far pointer to private data (passed to callback function)

Description:

This function registers an ART callback function. (see the callback functions at the end of this reference in the section ART Component Callback Functions Reference.) To unregister a function, pass NULL to the lpfnCallback parameter.

nCallbackID is a predefined value that identies the type of callback to register. The predefined constants for this variable reside in ARTAPI.H and begin with the prefix ART_CALLBACK_. Only one callback of each type may be registered at the same time.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- nCallbackID is not one of the predefined callback types.
- lpfnCallback is not NULL or a valid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */

AT_ERRCOUNT nErrcount; /* Tally of IG errors on  */
             /* the stack    */

VOID privateData; /* Any private data that you */
/* would like to pass to the */
/* callback   */

nErrcount = ART_callback_register(hIGear,
ART_CALLBACK_CREATE_MODIFY ,myCallbackFunc, (LPVOID)
&privateData);
```

ART_control_option_get

Group: Control

Declaration:

```
AT_ERRCOUNT ART_control_option_get(HIGEAR hIGear,
   AT_MODE nOption,
   const LPVOID lpValue
   );
```

Arguments:

hIGear	HIGEAR handle of an image.
nOption	An ART-defined control option constant .
lpValue	A far pointer to the return value.

Description:

This function returns a value of the control option specified by nOption parameters. See the function **ART_control_option_set()** for a list of the control option constants.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

ART_control_option_set

Group: Control

Declaration:

```
AT_ERRCOUNT ART_control_option_set(HIGEAR hIGear,
   AT_MODE nOption,
   const LPVOID lpValue
   );
```

Arguments:

hIGear	HIGEAR handle of an image.
nOption	An ART-defined constant for the Library option to change.
lpValue	A far pointer to a new value for the option.

Description:

This function modifies default library settings that changes the default behavior of the library. nOption specifies the option to set. The following list describes the available options and their default values:

Option	Type	Default	Description
ART_CONTROL_CLIP_TEXT	BOOL	TRUE	Text marks are clipped if they do not fit within the bounds of the annotation.

Option	Type	Default	Description
ART_CONTROL_CREATE_SCROLL	BOOL	FALSE	Does auto scrolling for the zoomed image when annotation is created. Applicable for Filled Rectangle Hollow Rectangle, Highlighter, Typed text, Attach-a-Note, RTF text, Text file, Text Stamp, Hot Spot, Redaction, Button, Encryption, Filled Ellipse and Hollow Ellipse marks.
ART_CONTROL_DISABLE_CLIPPING	BOOL	FALSE	Disables clipping
ART_CONTROL_DISABLE_PPM_FOR_FONTS	BOOL	FALSE	Disables PPM in font size calculations. If TRUE tells ImageGear to ignore the image resolution when displaying text annotations. See also description of ART_MARK_ATTRIBUTES structure and ART_DISABLE_PPM_FOR_FONTS_GET/ ART_DISABLE_PPM_FOR_FONTS_SET macros
ART_CONTROL_DO_NOT_MODIFY_CURSOR	BOOL	FASLE	Disables cursor modifications
ART_CONTROL_ENABLE_SELECTION_UNDO	BOOL	TRUE	If this option is set to FALSE, the selections are not placed to undo stack.
ART_CONTROL_FULL_TEXTMARKS_SELECT	BOOL	FALSE	If TRUE enables selecting a text mark using a mouse click when the text is not visible.
ART_CONTROL_LARGEFONTS_ENABLE	BOOL	FALSE	If TRUE then the use of very large fonts are enabled for text marks.
ART_CONTROL_LOAD_ORIENTATION	BOOL	TRUE	Enables loading an annotation with or without orientation
ART_CONTROL_PRINT_ENABLE	BOOL	TRUE	Enables printing of marks with the image
ART_CONTROL_RECT_SELECTION	BOOL	TRUE	Indicates the selection width of the rectangle around the mark
ART_CONTROL_RESIZE_ENABLE	BOOL	TRUE	This enables marks resizing while in the Edit Mode.
ART_CONTROL_SAVE_ENABLE	BOOL	TRUE	Enables saving of marks as an embeded file with a TIFF image

Option	Type	Default	Description
ART_CONTROL_SAVE_ORIENTATION	BOOL	TRUE	Enables saving an annotation with or without orientation
ART_CONTROL_SELECT_NO_CLIP	BOOL	FALSE	Ignores clipping during selection
ART_CONTROL_SELECT_SCROLL	BOOL	FALSE	Scroll and select at the same time
ART_CONTROL_SHADOW_FONTS	BOOL	FALSE	Enables shadow fonts
ART_CONTROL_SHOW_LOCKS	BOOL	FALSE	Whether to show padlocks on marks that have access locked
ART_CONTROL_SHOW_TEXT_WARNING	BOOL	FALSE	Whether to show warning text or not
ART_CONTROL_UNIX_LEGACY_FONTS_LOAD	BOOL	FALSE	Enables/disables legacy X font creation
ART_CONTROL_USE_ONE_AS_FALSE	BOOL	FALSE	Enables using 1 as FALSE value
ART_CONTROL_WANG_LOAD_COMPATIBILITY	BOOL	FALSE	Disables Wang compatibility for load
ART_CONTROL_WANG_SAVE_COMPATIBILITY	BOOL	FALSE	Disables Wang compatibility for save

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
nErrcount = ART_control_option_set(hIGear,
ART_CONTROL_SHOW_LOCKS, TRUE);
```

ART_edit_copy

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ART_edit_copy(HIGEAR hIGear,
   LPART_MARK_INDEX lpMarkIndex,
   DWORD dwMarkCount
```

```
);
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkIndex	A far pointer to an array list of mark identifiers
dwMarkCount	A variable of type DWORD which tells ART the number of marks to copy to the clipboard. This number indicates the number of mark indexes in the lpMarkIndex array.

Description:

This function copies an array of marks and places them on the clipboard.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- The memory pointed to by lpMarkIndex is invalid.
- One or more of the indexes in lpMarkIndex is not valid.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
ERRCOUNT nErrcount; /* tally of IG errors on the */
/* stack   */

ART_MARK_INDEX lpMarkIndex; /* number of mark(s) */
/* to copy toclipboard */

DWORD dwMarkCount; /* # of marks that have been */
/* selected  */

HGLOBAL hGlobal; /* Windows handle to memory */
/* area   */

. . .
case ID_EDIT_COPY:
nErrcount = ART_mark_select_count(hIGear,
   &dwMarkCount);
hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwMarkCount *
sizeof(ART_MARK_INDEX));
lpMarkIndex=(LPART_MARK_INDEX)GlobalLock(hGlobal);

nErrcount =
```

```
ART_mark_selected_first(hIGear,lpMarkIndex);
for (i = 1; i < (INT)dwMarkCount; i++)
{
lpMarkIndex[i] = lpMarkIndex[i - 1];
nErrcount = ART_mark_selected_next(hIGear,
lpMarkIndex + i);
}
nErrcount = ART_edit_copy(hIGear, lpMarkIndex,
dwMarkCount);

GlobalUnlock(hGlobal);
GlobalFree(hGlobal);
break;
```

ART_edit_cut

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ART_edit_cut(HIGEAR hIGear,
    LPART_MARK_INDEX lpMarkIndex,
    UINT nMarkCount
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkIndex	A far pointer to an array list of mark identifiers
nMarkCount	A variable of type UNIT which tells ART the number of marks in the list.

Description:

This function cuts an array list of marks and places it on the clipboard.

NOTE: The cut marks are visible until the screen is repainted.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- The memory pointed to by lpMarkIndex is invalid.
- One or more of the indexes in lpMarkIndex is not valid.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount; /* tally of IG errors on stack */
DWORD dwMarkCount; /* number of marks selected */
HGLOBAL hGlobal; /* Windows handle to  */
/* memory area  */
AT_MARK_INDEX lpMarkIndex; /* index to mark(s) */
HWND hWnd;  /* Windows handle to  */
/* image window  */

case ID_EDIT_CUT:
nErrcount = ART_mark_select_count(hIGear,
   &dwMarkCount);

hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwMarkCount *
sizeof(ART_MARK_INDEX));
lpMarkIndex = (LPART_MARK_INDEX)GlobalLock(hGlobal);

nErrcount =
ART_mark_selected_first(hIGear,lpMarkIndex);
for (i = 1; i < (INT)dwMarkCount; i++)
{
lpMarkIndex[i] = lpMarkIndex[i - 1];
nErrcount = ART_mark_selected_next(hIGear,
lpMarkIndex + i);
}

nErrcount = ART_edit_cut(hIGear, lpMarkIndex,
dwMarkCount);
nErrcount = ART_GUI_mark_paint(hIGear, IG_GRP_DEFAULT,
ART_INVALID_ID, hWnd, TRUE);

GlobalUnlock(hGlobal);
GlobalFree(hGlobal);
break;
```

ART_edit_paste

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ART_edit_paste(HIGEAR hIGear,
    HWND hWnd,
    DWORD dwGrpID
```

```
);
```

Arguments:

hIGear	HIGEAR handle to the image
hWnd	HWND handle to the window
dwGrpID	Identifier of group

Description:

This function pastes one or more marks from the clipboard to the current HIGEAR image. The marks are pasted on top of the image.

NOTE: The pasted marks are not visible until the screen is repainted.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount; /* tally of IG errors on stack */
DWORD dwMarkCount; /* # of marks available  */
/* for paste  */
HWND hWnd; /* Windows handle to  */
/* image window  */
DWORD dwGrpID;

nErrcount = ART_edit_paste_count(hIGear, &dwMarkCount);
if (dwMarkCount > 1)
{
dwGrpID = IG_GRP_DEFAULT;
nErrcount = ART_edit_paste(hIGear, hWnd, dwGrpID);
nErrcount = ART_GUI_mark_paint(hIGear, dwGrpID,
ART_INVALID_ID, hWnd, TRUE);
}
```

ART_edit_paste_count

Group: Clipboard

Declaration:

```
AT_ERRCOUNT ART_edit_paste_count(HIGEAR hIGear,
   LPDWORD lpMarkCount
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkCount	A far pointer which returns the number of mark items currently stored in the clipboard.

Description:

This function queries the clipboard for the number of marks available for pasting. The number of marks available for pasting from the clipboard is returned by `lpMarkCount`.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- The memory pointed to by `lpMarkCount` is invalid.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount; /* tally of IG errors on stack */
DWORD dwMarkCount; /* # of marks available  */
/* for paste  */

nErrcount = ART_edit_paste_count(hIGear, &dwMarkCount);
```

ART_edit_undo

Group: Undo

Declaration:

```
AT_ERRCOUNT ART_edit_undo(HIGEAR hIGear);
```

Arguments:

hIGear	HIGEAR handle to the image

Description:

This function reverses any changes made to any mark. Reversible operations include: creation, modification, selection, and deletions of marks. To clear the list of undos stored in memory, call **ART_edit_undo_clear()** function.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount; /* tally of IG errors on */
/* stack  */
HWND hWnd; /* Windows handle to */
/* image window  */

nErrcount = ART_edit_undo(hIGear);
nErrcount = ART_GUI_mark_paint(hIGear, IG_GRP_DEFAULT,
ART_INVALID_ID, hWnd, TRUE);
```

ART_edit_undo_clear

Group: Undo

Declaration:

```
AT_ERRCOUNT ART_edit_undo_clear(HIGEAR hIGear);
```

Arguments:

hIGear	HIGEAR handle to the image

Description:

This function clears all pending undos from memory. A subsequent call to **ART_edit_undo()** has no affect on the marks associated with the specified HIGEAR image.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
nErrcount = ART_edit_undo_clear(hIGear);
```

ART_edit_undo_count

Group: Undo

Declaration:

```
AT_ERRCOUNT ART_edit_undo_count(HIGEAR hIGear,
    LPDWORD lpUndoCount
    );
```

Arguments:

hIGear	`HIGEAR` handle to the image
lpUndoCount	A far pointer which returns the number of "undos" that can be performed

Description:

This function returns (in `lpUndoCount`) the number of times that `ART_edit_undo()` can be called.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- The memory pointed to by `lpUndoCount` is invalid.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image  */
AT_ERRCOUNT   nErrcount;    /* tally of IG errors on stack */
DWORD        dwMarkCount;  /* count of potential undos */
```

```
nErrcount = ART_edit_undo_count(hIGear, &dwMarkCount);
```

ART_edit_undo_enable

Group: Undo

Declaration:

```
AT_ERRCOUNT ART_edit_undo_enable(HIGEAR hIGear,
   BOOL bEnable
   );
```

Arguments:

hIGear	HIGEAR handle to the image
bEnable	A flag of type BOOL which allows you to enable or disable undo operations.

Description:

This function alllows you to enable or disable undo operations.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

ART_edit_undo_record

Group: Undo

Declaration:

```
AT_ERRCOUNT ART_edit_undo_record(HIGEAR hIGear,
   BOOL fRecord
   );
```

Arguments:

hIGear	HIGEAR handle to the image
fRecord	A flag of type BOOL which enables or disables the recording of every action, so that the actions can later be undone. If TRUE, every action is recorded. Set to FALSE when you want recording to stop.

Description:

This function enables or disables the recording of every action. If you call ART_edit_undo_record(), with fRecord set to TRUE, every action is recorded until this function is called again with fRecord set to FALSE. The next time **ART_edit_undo()** is called, all previously recorded actions become undone.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */

case ID_EDIT_SELECTALL:
ART_mark_select_count(hIGear, &dwSelectCount);
ART_mark_count(hIGear, &dwMarkCount);

/* if the number of marks equals the number of  */
/* selected marks, then we don't need to do this */
if (dwMarkCount != dwSelectCount)
{
ART_edit_undo_record(hIGear, TRUE);

ART_mark_first(hIGear, &hMarkIndex, NULL);
while (hMarkIndex != ART_INVALID_ID)
{
/* check to see if the mark is already */
/* selected */
ART_mark_is_selected(hIGear, hMarkIndex
&fSelected);
/* if not, then select it  */
if (!fSelected)
{
  ART_mark_select(hIGear, hMarkIndex, TRUE);
  ART_mark_next(hIGear, &hMarkIndex, NULL);
}
}
ART_edit_undo_record(hIGear, FALSE);

ART_GUI_mark_paint(hIGear, IG_GRP_DEFAULT,
```

```
ART_INVALID_ID, hWnd, TRUE);
}
break;
```

ART_enable_rotation_get

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_enable_rotation_get (HIGEAR hIGear,
    LPBOOL lpbEnable
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpbEnable	A pointer to BOOL value

Description:

This function determines whether rotation is possible for an annotation. A value of TRUE allows rotation, a FALSE value does not allow rotation. The default is TRUE.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the f unction call.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of Image */
AT_ERRCOUNT nErrCount; /* Tally of IG errors on stack */
BOOL bEnable;

nErrCount = ART_enable_rotation_get (hIGear,&bEnable);
```

ART_enable_rotation_set

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_enable_rotation_set (HIGEAR hIGear,
            BOOL bEnable);
```

Arguments:

hIGear	HIGEAR handle to the image
bEnable	TRUE/FALSE

Description:

This function determines whether rotation is possible or not for an annotation. A value of TRUE allows rotation, a FALSE value does not allow rotation. The default value is TRUE.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of Image */
AT_ERRCOUNT nErrCount; /* Tally of IG errors on stack */

nErrCount = ART_enable_rotation_set (hIGear,FALSE);
```

ART_file_append

Group: File

Declaration:

```
AT_ERRCOUNT ART_file_append(HIGEAR hIGear,
   HWND hWnd,
   DWORD dwGrpID,
   const LPSTR lpszFileName,
   DWORD dwPageNumber
   );
```

Arguments:

HIGear	HIGEAR handle to the image
HWnd	HWND handle of the image window
dwGrpID	Display group
lpszFileName	A far pointer to the name of the .ART file to append
dwPageNumber	Set this parameter to the page number to load if the .ART file has multiple pages. If the .ART file has one page, set this to 1.

Description:

This function appends the marks from a separate .ART file into the HIGEAR image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR            hIGear;         /* HIGEAR handle of image */
char FAR          *szARTFile;      /* Name of the ART file  */
/* to import  */
char FAR          *szARTAppend;     /* Name of the ART file  */
/* to append  */
AT_ERRCOUNT       nErrcount;     /* Tally of IG errors on stack
*/
HWND               hWnd;

nErrcount = IG_load_file("JoeFax.BMP", &hIGear);

nErrcount = ART_file_import(hIGear, szARTFile, 1);
nErrcount = ART_file_append(hIGear, hWnd, IG_GRP_DEFAULT,
szARTAppend, 1);
```

ART_file_export

Group: File

Declaration:

```
AT_ERRCOUNT ART_file_export(HIGEAR hIGear,
   const LPSTR     lpszFileName,
   DWORD           dwPageNumber,
   BOOL            fReplace
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpszFileName	A far pointer to a NULL-terminated string containing the name of the .ART file to export.
dwPageNumber	Use this parameter to save one page of an exisitng file to a new file. Set this parameter to the page number of the .ART file that you want to save to the new file.
fReplace	This parameter is relevant if the target .ART file already exists and has multiple pages. Set to TRUE to replace the page; set to FALSE to insert the page.

Description:

This function saves the marks associated with the HIGEAR image to a separate .ART file.

Other ways to save marks:

- Marks may be saved as an embedded file, but only if the format of the image file is TIFF - see **ART_control_option_set()**
- Marks may also be "burned in" permanently to an image using **ART_image_burn_in()**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szARTFile; /* Name of the ART file  */
/* to export  */
AT_ERRCOUNT nErrcount; /* Tally of IG errors on stack */

szARTFile = "ARTsnack.ART";
nErrcount = IG_save_file(hIGear, "Snack.png",
IG_SAVE_PNG);
nErrcount = ART_file_export(hIGear, szARTFile, 2,
FALSE);
```

ART_file_import

Group: File

Declaration:

```
AT_ERRCOUNT ART_file_import(HIGEAR hIGear,
```

```
HWND            hWnd,
DWORD           dwGrpID,
const LPSTR     lpszFileName,
DWORD           dwPageNumber
);
```

Arguments:

hIGear	HIGEAR handle to the image
hWnd	HWND handle to the window
dwGrpID	Identifier of group
lpszFileName	A far pointer to the name of the .ART file to import
dwPageNumber	Set this parameter to the page number to load if the .ART file has multiple pages. If the .ART file has one page, set this to 1.

Description:

This function loads the marks from a separate **.ART** file into the HIGEAR image. This is typically used for combining a newly loaded image with marks that were previously made and saved separately to an **.ART** file.

See also **ART_file_import_ex()** function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szARTFile; /* Name of the ART file  */
/* to import  */
AT_ERRCOUNT nErrcount; /* Tally of IG errors on stack */
HWND hWnd;
DWORD dwGrpID;

nErrcount = IG_load_file("JoeFax.BMP", &hIGear);
dwGrpID = IG_GRP_DEFAULT;
nErrcount = ART_file_import(hIGear, hWnd, dwGrpID,
szARTFile, 1);
```

ART_file_import_ex

Group: File

Declaration:

```
AT_ERRCOUNT ART_file_import_ex(HIGEAR hIGear,
    HWND            hWnd,
    DWORD           dwGrpID,
    const LPSTR     lpszFileName,
    DWORD           dwPageNumber,
    BOOL            bAppend
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hWnd	HWND handle to the window
dwGrpID	Identifier of group
lpszFileName	A far pointer to the name of the **.ART** file to import
dwPageNumber	Set this parameter to the page number to load if the **.ART** file has multiple pages. If the **.ART** file has one page, set this to 1.
bAppend	If this parameter is TRUE then current ART annotation will be simply added to the specified **.ART** file. Otherwise previous annotation in the file will be replaced by the current one.

Description:

This function loads the marks from a separate **.ART** file into the HIGEAR image. This is typically used for combining a newly loaded image with marks that were previously made and saved separately to an **.ART** file.

This function is the same as **ART_file_import()** but has supplementary parameter bAppend that allows you to determine if current ART annotation should be simply added to the specified .ART file (TRUE), or replace the existed one (FALSE).

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

ART_global_block_create

Group: Global Named Block Management

Declaration:

```
AT_ERRCOUNT ART_global_block_create(HIGEAR hIGear,
    const LPART_BLOCK_NAME lpBlockName,
    const LPVOID lpBlockData,
    DWORD dwBlockSize
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpBlockName	A far pointer to an 8 byte array of memory into which you set the type of block to create. The block type will be either an ART-defined constant beginning with ART_BLOCK_ or a user defined block. See ARTAPI.H for the full list of ART-defined constants.
lpBlockData	A far pointer of type VOID which points to the block data.
dwBlockSize	The size of lpBlockData, in bytes

Description:

This function creates a new global named block, or replaces an existing one. If a block of **type** lpBlockName exists, it is replaced by the new one. The block data pointed to by lpBlockData is copied into the new named block.

Unlike mark named blocks, which are associated with an individual mark, global named blocks are not associated with any mark - they act as default settings for the next created mark. When a new mark is created, some of its named blocks are created from the global named blocks.

NOTE: You may create your own global named blocks to store user-defined data.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- lpBlockName does not point to a valid address.
- lpBlockData does not point to a valid address.
- there is not enough memory to allocate dwBlockSize bytes of memory.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example

```
/* set the block size   */
dwBlockSize = sizeof(ART_ARROW) + sizeof(AT_POINT);

/* allocate memory for the arrow structure */
lpArrow = (LPART_ARROW)alloca((UINT)dwBlockSize);

/* initialize to zero   */
memset(lpArrow, 0, (UINT)dwBlockSize);

lpArrow->lArrowType  = ART_ARROW_POINTER;
lpArrow->lArrowEnd  = 0;
lpArrow->lArrowAngle = 15;
lpArrow->lArrowLength = 25;

/* create the named block and associate  */
/* it with this mark   */
ART_global_block_create(hIGear, ART_BLOCK_ANODAT,
lpArrow, dwBlockSize);

ART_GUI_tool_set (hIGear, &ma);
break;
```

ART_global_block_delete

Group: Global Named Block Management

Declaration:

```
AT_ERRCOUNT ART_global_block_delete(HIGEAR hIGear,
   const LPART_BLOCK_NAME lpBlockName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpBlockName	A far pointer to an 8-byte memory array contains the type of block to delete. The block type is either an ART-defined constant beginning with ART_BLOCK or a user-defined block. See ARTAPI.H for the full list of ART-defined constants.

Description:

This function deletes an existing global named block, lpBlockName, from the set of global named blocks. Reserved named blocks cannot be deleted. However, they may be modified by calling **ART_global_block_create()** function.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- lpBlockName does not point to a valid address.
- lpBlockName points to a reserved named block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */

nErrcount =
ART_global_block_delete(hIGear,"NOTES\000\000\000");
```

ART_global_block_first

Group: Global Named Block Management

Declaration:

```
AT_ERRCOUNT ART_global_block_first(HIGEAR hIGear,
    LPART_BLOCK_NAME lpBlockName
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpBlockName	A far pointer to an 8 byte array of memory that returns the type of block to query. The block type is either an ART-defined constant beginning with ART_BLOCK or a user defined block. See ARTAPI.H for the full list of ART-defined constants.

Description:

This function gets the name of the first global block. The block data for the named block can be retrieved by calling **ART_global_block_query()** function.

Call **ART_global_block_next()** function to get the block name of the named block that follows the one that was passed into lpBlockName. This function sets lpBlockName to zeroes if there are no global blocks. Block names are returned in no particular order.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- lpBlockName does not point to a valid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example

```
HIGEAR  hIGear; /* HIGEAR handle of image */
ART_BLOCK_NAME lpBlockName; /* returned type of first */
/* global block  */
AT_ERRCOUNT nErrcount; /* tally of IG errors on */
/* stack  */

nErrcount = ART_global_block_first(hIGear,&lpBlockName);
```

ART_global_block_next

Group: Global Named Block Management

Declaration:

```
AT_ERRCOUNT ART_global_block_next(HIGEAR hIGear,
   LPART_BLOCK_NAME lpBlockName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpBlockName	A far pointer to an 8-byte array of memory that returns the type of block to query. The block type is either an ART-defined constant beginning with ART_BLOCK or a user-defined block. See ARTAPI.H file for the full list of ART-defined constants.

Description:

This function gets the name of the next global block associated with the HIGEAR image, and iterates through all global named blocks. .**ART_global_block_first()** gets the first block name. The block data for the named block is retrieved by calling **ART_global_block_query()**function. This function sets lpBlockName to zero when there are no more blocks, providing you with an easy check for ending your loop. Block names are returned in no particular order.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- lpBlockName does not point to a valid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
ART_BLOCK_NAME lpBlockName; /* returned type of first */
/* global block  */
AT_ERRCOUNT nErrcount; /* tally of IG errors */
/* on stack  */

nErrcount = ART_global_block_first(hIGear,&lpBlockName);
while (*lpBlockName)
{
  if (!memcmp(&lpBlockName, "EYEBALL\0", 8))
break;

nErrcount = ART_global_block_next(hIGear,
&lpBlockName);
}
```

ART_global_block_query

Group: Global Named Block Management

Declaration:

```
AT_ERRCOUNT ART_global_block_query(HIGEAR hIGear,
    const LPART_BLOCK_NAME lpBlockName,
    LPVOID lpBlockData,
    LPDWORD lpBlockSize
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpBlockName	A far pointer to an 8 byte array of memory that specifies the type of block to query. The block type is either an ART-defined constant beginning with ART_BLOCK or a user-defined block. See ARTAPI.H file for the full list of ART-defined constants.
lpBlockData	A far VOID pointer which returns block data.
lpBlockSize	The size of lpBlockData (in bytes).

Description:

This function queries a global named block for its data returning it in lpBlockData. The buffer pointed to by lpBlockData must be large enough to fit the named block data. The size of the block is returned by lpBlockSize.

To find the size of a buffer necessary to hold the block data, call ART_global_block_query() with lpBlockData set to NULL. lpBlockSize returns the size of the block.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- lpBlockName does not point to a valid address.
- lpBlockData does not point to a valid address or is not **NULL**.
- lpBlockData points to a buffer smaller than the block data.
- lpBlockSize points to an invalid address.
- lpBlockSize points to a value smaller than the size of the block data.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
LPVOID blockData; /* data in global block  */
DWORD blockSize; /* size of block, in bytes */
AT_ERRCOUNT nErrCount; /* number of errors on the */
/* error stack  */
/* get the block size */

nErrCount = ART_global_block_query(hIGear,
ART_BLOCK_GROUP, NULL, &blockSize);

blockData = (LPVOID)alloca(blockSize);

/* but the data into blockData */
nErrCount = ART_global_block_query(hIGear,
ART_BLOCK_GROUP, blockData, &blockSize);
```

ART_group_access

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_access(HIGEAR hIGear,
    const LPSTR szGroupName,
    BOOL fEnable
    );
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	A far pointer to a null-terminated string that tells ART which group of marks' access to change.
fEnable	A flag of type BOOL that tells ART whether to allow editing for all the marks in the group specified by szGroupName.

Description:

This function sets the access control (editable or view only) for all of the marks in the group specified by szGroupName. When fEnable is set to TRUE, the end user can edit or modify marks in the group; when set to FALSE the end user cannot edit or modify marks in the group.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- szGroupName does not point to a valid address.
- No group matches szGroupName.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* type of group to make  */
/* accessible  */

/* Deny access to the MIS Group */
szGroupName = "MIS Group";

nErrcount = ART_group_access(hIGear,szGroupName,FALSE);
```

Sample: **None**

ART_group_bring_to_front

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_bring_to_front(HIGEAR hIGear,
   const LPSTR szGroupName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	Far pointer to a NULL-terminated string containing the mark group name

Description:

This function moves a group of marks to the front of the stacking order.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- No group matches szGroupName.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* Name of group to */
/* bring to front  */

nErrcount = ART_group_bring_to_front(hIGear,szGroupName);
```

Sample: None

ART_group_delete

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_delete(HIGEAR hIGear,
   const LPSTR szGroupName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	A far pointer to a NULL-terminated string that contains the name of the group that contains the marks for deletion.

Description:

This function deletes any marks that belong to the mark group.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- `szGroupName` does not point to a valid address.
- No group matches `szGroupName`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* name of group to delete */

szGroupName = "Sticky Notes Group";

nErrcount = ART_group_delete(hIGear, szGroupName);
```

Sample: None

ART_group_edit_copy

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_edit_copy(HIGEAR hIGear,
   const LPSTR szGroupName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	A far pointer to a NULL-terminated string that contains the group of marks to copy to the clipboard

Description:

This function copies a group of marks to the clipboard. Use **szGroupName** to tell ART which group of marks to copy.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- No group matches szGroupName.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* name of group from */
/* which to copy mark */
AT_ERRCOUNT nErrcount; /* tally of IG errors */
/* on stack  */

/* Copy the Sticky Notes Group to the clipboard */
szGroupName = "Sticky Notes Group";

nErrcount = ART_group_edit_copy(hIGear, szGroupName);
```

Sample: None

ART_group_edit_cut

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_edit_cut(HIGEAR hIGear,
   const LPSTR szGroupName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	A far pointer to a null-terminated string that tells ART which group of marks to cut from the HIGEAR image and copy to the clipboard

Description:

This function cuts a group of marks from the HIGEAR image and copies them to the clipboard.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- No group matches szGroupName.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* name of group from  */
/* which to cut marks */
AT_ERRCOUNT nErrcount; /* tally of IG errors */
/* on stack  */

szGroupName = "Time Group";

nErrcount = ART_group_edit_cut(hIGear, szGroupName);
```

Sample: None

ART_group_select

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_select(HIGEAR hIGear,
   const LPART_GROUP_NAME szGroupName,
   BOOL fSelect
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpGroupName	A far pointer to a NULL-terminated string that identifies the name of the mark group for selection.
fSelect	A flag of type BOOL that tells ART whether to select or deselect the marks in the specified group. If set to TRUE, the marks in the group are selected.

Description:

This function selects or deselects all marks that belong to the specified mark group. When `fSelect` is set to `TRUE`, all marks in the group are selected. When `fSelect` is set to `FALSE`, all marks in the group are deselected.

An error is set set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- `szGroupName` does not point to a valid address.
- No group matches `szGroupName`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* Type of Group to */
/* select all marks from */

szGroupName = "Former Employee Group";
nErrcount = ART_group_select(hIGear, szGroupName, TRUE);
```

Sample: None

ART_group_send_to_back

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_send_to_back (HIGEAR hIGear,
   const LPSTR szGroupName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	A far pointer to a NULL-terminated string that contains the name of the mark group.

Description:

This function moves a group of marks to the back of the stacking order. An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- No group matches szGroupName.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* Name of group to bring */
/* to front  */

nErrcount = ART_group_send_to_back(hIGear, szGroupName);
```

Sample: None

ART_group_visible

Group: Mark Grouping

Declaration:

```
AT_ERRCOUNT ART_group_visible(HIGEAR hIGear,
```

```
const LPSTR szGroupName,
BOOL fVisible
);
```

Arguments:

hIGear	HIGEAR handle to the image
szGroupName	A far pointer to a NULL-terminated string that identifies the name of the mark group to make visible
fVisible	A flag of type BOOL that tells ART whether to make all of the marks in the group visible. If set to TRUE, all marks become visible.

Description:

This function sets the visibility (hide or show) of all of the marks in the group identified by `szGroupName`. The `fVisible` field of each of the mark's attribute structures will be set to `TRUE` or `FALSE`, depending on the `fVisible` parameter.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- `szGroupName` does not point to a valid address.
- No group matches `szGroupName`.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
char FAR *szGroupName; /* type of group of marks */
/* to make visible  */
HIGEAR hIGear; /* HIGEAR handle of image */
LPSTR szGroupName; /* data in global block  */
DWORD dwBlockSize; /* size of block, in bytes */
AT_ERRCOUNT nErrCount; /* number of errors on the */
/* error stack  */

nErrCount = ART_global_block_query(hIGear,
ART_BLOCK_GROUP, NULL, &dwBlockSize);

szGroupName = (LPSTR)alloca(dwBlockSize);

nErrCount = ART_global_block_query(hIGear, ART_BLOCK_GROUP,
(LPVOID)szGroupName, &dwBlockSize);

nErrcount = ART_group_visible(hIGear,szGroupName,TRUE);
```

Sample: None

ART_image_burn_in

Group: Image Utility

Declaration:

```
AT_ERRCOUNT ART_image_burn_in(HIGEAR hIGear,
    HWND hWnd,
    DWORD dwGrpID,
    AT_MODE fOptions,
    const LPSTR szGroup,
    PART_MARK_INDEX lpMarkIndex,
    UINT nMarkCount
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hWnd	HWND handle to the window
dwGrpID	Identifier of group
fOptions	An ART-defined constant (a bit flag) that tells what type of marks to burn in to the HIGEAR image. These are defined in ARTAPI.H, and are shown below.
szGroup	A far pointer to a null-terminated string that tells ART what group of marks to merge, when the ART_BURN_IN_GROUP bit is set.
lpMarkIndex	A far pointer to the mark identifier, used when the ART_BURN_IN_LIST bit is set.
hMarkCount	The count of marks or groups, ignored unless the ART_BURN_IN_LIST bit is set.

Description:

This function merges annotation marks permanently into an image. The fOptions parameter specifies which marks are merged.

fOptions may be a bitwise combination of any of the following values:

ART_BURN_IN_ALL	Burns in all annotation marks
ART_BURN_IN_DELETE_MARKS	Deletes affected marks after burning them in.
ART_BURN_IN_GROUP	Burns in a group of marks

ART_BURN_IN_LIST	Burns in a list of marks
ART_BURN_IN_VISIBLE	Burns in visible marks
ART_BURN_IN_SELECTED	Burns in selected marks
ART_BURN_IN_AUTO_PROMOTE	Automatically promotes image
ART_BURN_IN_NO_PROMOTE	No promotion before burn-in

The szGroup parameter is ignored, unless **fOptions** includes ART_BURN_IN_GROUP. The lpMarkIndex parameter is ignored, unless fOptions includes ART_BURN_IN_LIST. nMarkCount is the count of items in the list pointed to by lpMarkIndex.

To restrict the number of marks that are burned into the image, set more options. For example, if ART_BURN_IN_GROUP, ART_BURN_IN_SELECTED and ART_BURN_IN_VISIBLE are set, only those marks that are in the group szGroup, that are selected and visible, are burned into the image.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- fOptions includes ART_BURN_IN_LIST and the memory pointed to by lpMarkIndex is invalid.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
HWND hWnd;
DWORD dwGrpID;

/* Burn in all visible marks in the group  */
/* "Unclassified"     */
dwGrpID = IG_GRP_DEFAULT;
ART_image_burn_in(hIGear, hWnd, dwGrpID,
ART_BURN_IN_VISIBLE|ART_BURN_IN_GROUP,
"Unclassified", NULL, 0L);
```

ART_mark_block_create

Group: Named Block Management

Declaration:

```
AT_ERRCOUNT ART_mark_block_create(HIGEAR hIGear,
```

```
ART_MARK_INDEX hMarkIndex,
const LPART_BLOCK_NAME lpBlockName,
LPVOID lpBlockData,
DWORD dwBlockSize
);
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpBlockName	Tells ART what type of block to create. Use an ART-defined constant beginning with ART_BLOCK. See ARTAPI.H for the full list. This parameter is also a user-defined block type.
lpBlockData	A far pointer to the Block to copy into a new named block
dwBlockSize	Size of lpBlockdata (in bytes)

Description:

This function creates or replaces a named block and associates a new block with the annotation mark pointed to by hMarkIndex. If a block of type lpBlockName exists, the new one replaces the previous one. The block data pointed to by lpBlockData is copied into the new named block.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- lpBlockName does not point to a valid address.
- lpBlockData does not point to a valid address.
- There is not enough memory to allocate dwBlockSize bytes of memory.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image  */
AT_MARK_ATTRIBUTES ma; /* mark attributes structure */
ART_POINTS lpPoints; /* screen coordinates   */
ART_MARK_INDEX hMarkIndex; /* index to mark getting  */
/* new block   */
DWORD  dwBlockSize; /* size in bytes of block */

/* set the block size */
dwBlockSize = sizeof(ART_POINTS) + sizeof(AT_POINT);
```

```
/* allocate memory for the points structure */
lpPoints = (LPART_POINTS)LocalAlloc(LPTR,
(UINT)dwBlockSize);

/* initialize to zero */
memset(lpPoints, 0, (UINT)dwBlockSize);

/* fill the structure */
lpPoints->lMaxPoints    = 2;
lpPoints->lPoints       = 2;

lpPoints->ptPoint[0].x  = (ma.rcBounds.right –
ma.rcBounds.left) - 1;
lpPoints->ptPoint[0].y  = 0;
lpPoints->ptPoint[1].x  = 0;
lpPoints->ptPoint[1].y  = (ma.rcBounds.bottom –
ma.rcBounds.top) - 1;

/* create the named block and associate it with this  */
/* mark         */
nErrcount = ART_mark_block_create(hIGear, hMarkIndex,
ART_BLOCK_ANODAT, lpPoints, dwBlockSize);
```

ART_mark_block_create_ex

Group: Named Block Management

Declaration:

```
AT_ERRCOUNT ART_mark_block_create_ex(HIGEAR hIGear,
   HWND hWnd,
   DWORD dwGrpID,
   ART_MARK_INDEX hMarkIndex,
   const LPART_BLOCK_NAME lpBlockName,
   LPVOID lpBlockData,
   DWORD dwBlockSize
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hWnd	HWND handle to the window
dwGrpID	Identifier of group
hMarkIndex	Mark identifier
lpBlockName	Tells ART what type of block to create. Use an ART-defined constant beginning with ART_BLOCK. See ARTAPI.H for the full list. This parameter is also a user-defined block type.
lpBlockData	A far pointer to the Block to copy into a new named block
dwBlockSize	Size of lpBlockdata, in bytes

Description:

This function creates or replaces a named block and associates a new block with the annotation mark pointed to by hMarkIndex. If a block of type lpBlockName exists, the new one replaces the previous one. The block data pointed to by lpBlockData is copied into the new named block.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- lpBlockName does not point to a valid address.
- lpBlockData does not point to a valid address.
- There is not enough memory to allocate dwBlockSize bytes of memory.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image  */
AT_MARK_ATTRIBUTES ma; /* mark attributes structure */
ART_POINTS lpPoints; /* screen coordinates   */
ART_MARK_INDEX hMarkIndex; /* index to mark getting  */
/* new block   */
DWORD  dwBlockSize; /* size in bytes of block */
HWND   hWnd;
DWORD  dwGrpID;

/* set the block size */
dwBlockSize = sizeof(ART_POINTS) + sizeof(AT_POINT);
```

```
/* allocate memory for the points structure */
lpPoints = (LPART_POINTS)LocalAlloc(LPTR,
(UINT)dwBlockSize);

/* initialize to zero */
memset(lpPoints, 0, (UINT)dwBlockSize);

/* fill the structure */
lpPoints->lMaxPoints    = 2;
lpPoints->lPoints       = 2;

lpPoints->ptPoint[0].x  = (ma.rcBounds.right –
ma.rcBounds.left) - 1;
lpPoints->ptPoint[0].y  = 0;
lpPoints->ptPoint[1].x  = 0;
lpPoints->ptPoint[1].y  = (ma.rcBounds.bottom –
ma.rcBounds.top) - 1;

/* create the named block and associate it with this  */
/* mark         */
dwGrpID = IG_GRP_DEFAULT;
nErrcount = ART_mark_block_create_ex(hIGear, hWnd, dwGrpID,
hMarkIndex, ART_BLOCK_ANODAT, lpPoints, dwBlockSize);
```

ART_mark_block_delete

Group: Named Block Management

Declaration:

```
AT_ERRCOUNT ART_mark_block_delete(HIGEAR hIGear,
   ART_MARK_INDEX hMarkIndex,
   const LPART_BLOCK_NAME lpBlockName
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpBlockName	Tells ART what type of block to delete. Use an ART-defined constant beginning with ART_BLOCK. See ARTAPI.H for the full list. This parameter is also a user-defined block type.

Description:

This function deletes an existing named block. It deletes the named block lpBlockName from the mark referenced by hMarkIndex.

- An error will be set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- `lpBlockName` does not point to a valid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image */
ART_MARK_INDEX hMarkIndex; /* index to the mark the  */
/* block belongs to  */

nErrcount = ART_mark_block_delete(hIGear, hMarkIndex,
ART_BLOCK_ANODAT);
```

ART_mark_block_first

Group: Named Block Management

Declaration:

```
AT_ERRCOUNT ART_mark_block_first(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    LPART_BLOCK_NAME lpBlockName
    );
```

Arguments:

hIGear	`HIGEAR` handle to the image
hMarkIndex	Mark identifier
lpBlockName	A far pointer to an 8-byte array of memory that returns the type of block of the mark referenced by `hMarkIndex`. The block type is either an ART-defined constant beginning with `ART_BLOCK` or a user-defined block. See `ARTAPI.H` for the full list of ART-defined constants.

Description:

This function gets the first block name. The block data for the named block retrieved by calling **`ART_mark_block_query()`**. **`ART_mark_block_next()`** is called to get the name of the next named block. If there are no blocks, `ART_mark_block_first()` sets `lpBlockName` to zero.

TIP: Many of ART's callback functions include the index number as a parameter. As part of your callback function, you can call `ART_mark_block_first()` and **`ART_mark_block_next()`** using the mark index that was passed in.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle. There are no marks with a matching index block.
- `lpBlockName` does not point to a valid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image */
ART_MARK_INDEX hMarkIndex; /* index to mark   */
ART_BLOCK_NAME lpBlockname; /* type of block   */
/* to return   */
CHAR szFileName[_MAX_PATH]; /* file name buffer  */
DWORD dwBlockSize;  /* size of a named  Block  */

nErrcount = ART_mark_block_first(hIGear, hMarkIndex,
&lpBlockName);

while (*lpBlockName)
{
if (!memcmp(lpBlockName, ART_BLOCK_FILENAM,
sizeof(ART_BLOCK_NAME))
{
nErrcount = ART_mark_block_query(hIGear,
hMarkIndex, &lpBlockName,
(LPVOID)szFileName, &dwBlockSize);
if (!strcmp(szFileName, "Clown.tif"))
{
nErrcount = RT_mark_send_to_back(hIGear,
hMarkIndex);
}
}
nErrcount = ART_mark_block_next(hIGear, hMarkIndex,
&lpBlockName);
}
```

ART_mark_block_next

Group: Named Block Management

Declaration:

```
AT_ERRCOUNT ART_mark_block_next(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    LPART_BLOCK_NAME lpBlockName
```

```
);
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpBlockName	A far pointer to an 8-byte array of memory that returns the type of block of the mark referenced by hMarkIndex. The block type is either an ART-defined constant beginning with ART_BLOCK or a user-defined block. See ARTAPI.H for the full list of ART-defined constants.

Description:

This function gets the block name of the named block that follows the one that was last passed into lpBlockName. **ART_mark_block_first()** is called to get the first block name. The block data for the named block is retrieved by calling **ART_mark_block_query()**. Both ART_mark_block_next() and **ART_mark_block_first()** set lpBlockName to zero when there are no more blocks (providing you an easy way to know when to end your loop). Block names are returned in no particular order.

NOTE: To get the mark index needed for this function, you can call any of ART's functions that return a mark index, for example: **ART_mark_first()**.

TIP: Many of ART's callback functions include the index number as a parameter. As part of your callback function, you can call **ART_mark_block_first()** and ART_mark_block_next() using the passed mark index.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- lpBlockName does not point to a valid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

See example under **ART_mark_block_first()**function.

ART_mark_block_query

Group: Named Block Management

Declaration:

```
AT_ERRCOUNT ART_mark_block_query(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    const LPART_BLOCK_NAME lpBlockName,
    LPVOID lpBlockData,
    LPDWORD lpBlockSize
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpBlockName	Tells ART what type of block to query. Use an ART-defined constant beginning with ART_BLOCK. See ARTAPI.H for the full list. This parameter is also a user-defined block type.
lpBlockData	The block data returned
lpBlockSize	The size of BlockData (in bytes)

Description:

This function queries a mark for the data from one of its named blocks. The block data for the named block associated with lpBlockName is copied into lpBlockData. The buffer pointed to by lpBlockData is large enough to fit the named block data. The size of the block is returned to lpBlockSize.

To find the size needed for a buffer to hold the block data, set lpBlockData to NULL and the size of the block is returned in lpBlockSize.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- lpBlockName does not point to a valid address.
- lpBlockData does not point to a valid address or is not NULL.
- lpBlockData points to a buffer smaller than the block data.
- lpBlockSize points to an invalid address.
- lpBlockSize points to a value smaller than the size of the block data.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image */
ART_MARK_INDEX hMarkIndex; /* index to mark   */
/* associated   */
/* with block being  */
/* queried    */
LPVOID   BlockData; /* memory to hold data  */
/* from block  */
DWORD  BlockSize; /* size of blocks,   */
/* in byte    */

nErrcount = ART_mark_block_query(hIGear, hMarkIndex,
ART_BLOCK_ANODAT, NULL, &BlockSize);

BlockData = (LPVOID)alloca(BlockSize);

nErrcount = ART_mark_block_query(hIGear, hMarkIndex,
ART_BLOCK_ANODAT, &BlockData, &BlockSize);
```

ART_mark_bring_to_front

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_bring_to_front(HIGEAR hIGear,
   ART_MARK_INDEX hMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier

Description:

This function moves a mark to the front of the stacking order. To move a group of marks to the front, use `ART_group_bring_to_front()`.

NOTE: The mark must be repainted in order to see a change in the stacking order.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT  nErrcount; /* tally of IG errors  */
/* on stack    */
ART_MARK_INDEX hMarkIndex /* index for mark to  */
/* bring to front   */

nErrcount = ART_mark_selected_first(hIGear,
&hMarkIndex);

if (hMarkIndex != ART_MARK_INVALID)
{
nErrcount = ART_mark_bring_to_front(hIGear,
hMarkIndex);
}
```

ART_mark_count

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_count(HIGEAR hIGear,
   LPDWORD lpMarkCount
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkCount	A far pointer that returns the total number of marks associated with the HIGEAR image.

Description:

This function returns the total number of marks associated with the current HIGEAR image.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR   hIGear;
DWORD   dwMarkCount;
nErrcount = ART_mark_count(hIGear, &dwMarkCount);
```

ART_mark_create

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_create(HIGEAR hIGear,
   const LPART_MARK_ATTRIBUTES lpMarkAttr,
   LPART_MARK_INDEX lpMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkAttr	Mark Description
lpMarkIndex	Mark Identifier

Description:

This function creates a new mark. Marks are initially created with two reserved named blocks: index and group. The index block is assigned a unique number, starting at 0. The index is taken from the global index block, then the global index block is incremented by one. The group is taken from the global group block. The following marks require additional named blocks to be created: lines, images, arrows, buttons, audio and text.

Returned in `lpMarkIndex` is the value stored in the index named block. The fields in the `lpMarkAttr` parameter set the attributes of the new mark, andare filled with default values. Call **`ART_mark_modify()`** to modify the attributes of the mark.

If the `dwTime` field is set to 0, the library is set to the current time. Not all fields are used for each mark type. An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- `lpMarkAttr` is an invalid address or has one or more invalid fields.
- The memory pointed to by `lpMarkIndex` is read-only, invalid or not large enough.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image */
AT_MARK_ATTRIBUTES ma;  /* attributes structure */
AT_MARK_INDEX hMarkIndex; /* index to the mark just */
/* created   */

/* set the mark type */
ma.dwType = ART_MARK_TEXT_STAMP;

/* set the mark color */
ma.rgbColor1.rgbRed = 0xFF;

/* set the mark to be visible */
ma.bVisible = TRUE;

/* create the mark */
nErrcount = ART_mark_create(hIGear, &ma, &hMarkIndex);
if (nErrcount != 0)
{
   ErrorReport(hWnd);
}
else
{
    ART_GUI_mark_paint(hIGear, IG_GRP_DEFAULT, hMarkIndex,
hWnd, TRUE);
}
```

ART_mark_delete

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_delete(HIGEAR hIGear,
   ART_MARK_INDEX hMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark Identifier

Description:

This function deletes the mark whose index block value is the same as `hMarkIndex`.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- There are no marks with a matching index block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image */
ART_MARK_INDEX hMarkDelete, hMarkIndex; /* Mark index  */
   /* numbers */

ART_mark_selected_first(hIGear, &hMarkIndex);
if (hMarkIndex != ART_MARK_INVALID)
{
hMarkDelete = hMarkIndex;
/* Set mark to be deleted to  */
   /* current index number   */
ART_mark_delete(hIGear, hMarkDelete);
}
```

ART_mark_first

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_first(HIGEAR hIGear,
    LPART_MARK_INDEX lpMarkIndex,
    LPART_MARK_ATTRIBUTES lpMarkAttr
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkIndex	Mark identifier returned
lpMarkAttr	Mark description returned or NULL

Description:

This function returns the mark index of the first mark associated with the HIGEAR image. The attributes for the mark are copied into lpMarkAttr. The lpMarkAttr parameter is set to NULL if the mark attributes are not needed. **ART_mark_next()** gets the mark index and attributes of the mark that follows the one that was passed into lpMarkIndex. These functions set lpMarkIndex to ART_INVALID_ID when there are no more marks to iterate through. Mark indexes are returned in the same order as the stacking order.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- The memory pointed to by lpMarkAttr is read-only (or not NULL), invalid or not large enough.
- The memory pointed to by lpMarkIndex is read-only, invalid or not large enough.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
BOOL fSelected; /* flag for whether mark has been
selected */
AT_MARK_INDEX  hMarkIndex; /* index number for marks */

/* Get index of first mark created */
ART_mark_first(hIGear, &hMarkIndex, NULL);

/* While there are valid index number, iterate through them and
select all of those marks which have not yet been selected */
while (hMarkIndex != ART_INVALID_ID)
{
/* check to see if the mark is already selected */
ART_mark_is_selected(hIGear, hMarkIndex, &fSelected);

/* if not, then select it */
if (!fSelected)
{
ART_mark_select(hIGear, hMarkIndex, TRUE);
```

```
ART_mark_next(hIGear, &hMarkIndex, NULL);
}
}
```

ART_mark_is_selected

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_is_selected(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    LPBOOL lpSelected
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpSelected	A far pointer that returns the current setting of the selection field in the mark's attribute structure. The value of this field is Boolean.

Description:

This function returns the current selection state of a mark. If the mark is selected, lpSelected returns a value of TRUE. If the mark is not selected, lpSelected returns a value of FALSE.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block. lpSelected points to an invalid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

See example under **ART_mark_first()**

ART_mark_is_visible

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_is_visible(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    BOOL lpVisible
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpVisible	A far pointer that returns the current setting of the visibility field in the mark's attribute structure. The value of this field is Boolean.

Description:

This function returns the current visibility state of a mark. The value returned in lpVisible is the value of the bVisible field of the mark's attribute structure.

An error is set if any of the following conditions are met:

- **hIGear** does not reference a valid ImageGear handle. There are no marks with a matching index block.
- **lpVisible** points to an invalid address.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

See example under **ART_mark_visible()** function.

ART_mark_modify

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_modify(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    LPART_MARK_ATTRIBUTES lpMarkAttr
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpMarkAttr	A far pointer to a structure of type ART_MARK_ATTRIBUTES that holds all settings for the mark's attributes

Description:

This function changes the attributes of a mark. The attributes are stored in a structure of type ART_MARK_ATTRIBUTES, that is defined in ARTAPI.H. You can set any number of the attributes of this structure and then pass it to this function.

NOTE: The mark is repainted in order to see the change in it's attributes.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- lpMarkAttr is an invalid address or has one or more invalid fields.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR  hIGear;  /* HIGEAR handle of image */
AT_MARK_ATTRIBUTES ma;  /* mark attributes */
/* structure */
AT_MARK_INDEX markIndex /* index to the mark */

ART_mark_selected_first(hIGear, &hMarkIndex);

if (hMarkIndex != ART_MARK_INVALID)
{
ART_mark_query(hIGear, markIndex, &ma);
ma.dwType = 506;  /* arrow    */
ma.dwLineSize = 5;  /* line thickness =  */
/* 5 pixels    */

nErrcount = ART_mark_modify(hIGear, markIndex, &ma);
}
```

ART_mark_next

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_next(HIGEAR hIGear,
    LPART_MARK_INDEX lpMarkIndex,
    LPART_MARK_ATTRIBUTES lpMarkAttr
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkIndex	Mark identifier
lpMarkAttr	Mark description

Description:

This function is called after **ART_mark_first()** to get the mark index and attributes of the next mark. The attributes for the mark are copied into lpMarkAttr. The lpMarkAttr parameteris set to NULL if the mark attributes are not needed. Both this function and **ART_mark_first()** set lpMarkIndex to ART_INVALID_MARK when there are no more marks to iterate through. Mark indexes are returned in the same order as the stacking order.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- The memory pointed to by lpMarkAttr is read-only (or not NULL), invalid or not large enough.
- The memory pointed to by lpMarkIndex is read-only, invalid or not large enough.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

See example under **ART_mark_first()**function.

ART_mark_query

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_query(HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    LPART_MARK_ATTRIBUTES lpMarkAttr
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
lpMarkAttr	Mark description

Description:

This function queries a mark for its attributes that are stored in the structure AT_MARK_ATTRIBUTES. The attributes for the mark associated with hMarkIndex are copied into lpMarkAttr, and can then be examined. To bring a particular mark to the front of the stacking order, use **ART_mark_bring_to_front()**.

An error is set if any of the following conditions are met:

- hIGear does not reference a valid ImageGear handle.
- There are no marks with a matching index block.
- The memory pointed to by lpMarkAttr is read-only, invalid or not large enough.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR  hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT  nErrcount; /* tally of IG errors on  */
/* the stack   */
ART_MARK_INDEX hMarkIndex; /* index to the mark  */
/* being queried  */
ART_MARK_ATTRIBUTES ma ; /* structure holding the  */
/* mark's attributes  */
UINT arrowCount;  /* number of arrow marks  */
/* assoc. with image  */

/* Get index of first mark created */
```

```
ART_mark_first(hIGear, &hMarkIndex, NULL);

/* While there are valid index number, iterate through them and
keep count of any arrow marks associated with the image */
while (hMarkIndex != ART_INVALID_ID)
{
nErrcount = ART_mark_query(hIGear, hMarkIndex, &ma);
if (ma.dwType = 506)
{
arrowCount++
}
   nErrcount = ART_mark_next(hIGear,&HMarkIndex,NULL);
}
```

ART_mark_rotate

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_rotate (HIGEAR hIGear,
    ART_MARK_INDEX hMarkIndex,
    DOUBLE dblRotate
    );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
dblRotate	Rotation angle

Description:

This function allows individual mark rotation. Currently only image marks (embedded and image reference) can be rotated.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- There are no marks with a matching index block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR              hIGear;       /* HIGEAR handle of image  */
```

```
AT_ERRCOUNT          nErrcount;     /* tally of IG errors on  */
/* stack    */
ART_MARK_INDEX       hMarkIndex;    /* index to current mark  */
ART_MARK_ATTRIBUTES ma;    /* structures for  */
/* attributes of current  */
/* mark  */

nErrcount = ART_mark_query(hIGear, hMarkIndex, &ma);

if ((ma.dwType == ART_MARK_IMAGE_EMBEDDED) ||
   (ma.dwType == ART_MARK_IMAGE_REFERENCE))
{      /* if mark is an embedded image or image reference*/
       nErrcount = ART_mark_rotate(hIGear, hMarkIndex, 45.0);
}
```

ART_mark_select

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_select(HIGEAR hIGear,
   ART_MARK_INDEX hMarkIndex,
   BOOL fSelect
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
fSelect	If set to FALSE, the mark is deselected; if TRUE, the mark is selected.

Description:

This function selects or deselects a mark. When `fSelect` is set to `TRUE`, the mark is selected. When `fSelect` is set to `FALSE`, the mark is deselected.

NOTE: The mark must be repainted in order to see a change in the selection state.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- There are no marks with a matching index block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

See example under **ART_mark_first()** function.

ART_mark_select_count

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_select_count(HIGEAR hGear,
   LPWORD lpSelected
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpSelected	A far DWORD pointer that returns the number of selected marks.

Description:

This function counts the number of marks selected by the user or selected by calling **ART_mark_select()** function.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image  */
AT_ERRCOUNT nErrcount; /* tally of IG errors on   */
/* stack     */
char szBuf[100]; /* Temporary string buffer */

nErrcount = ART_mark_select_count(hIGear, &dwMarkCount);
wsprintf(szBuf, _TEXT("There are %ld marks selected."),
dwMarkCount);
MessageBox(HWND_DESKTOP,szBuf,NULL,MB_ICONINFORMATION);
```

ART_mark_selected_first

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_selected_first(HIGEAR hGear,
   LPART_MARK_INDEX lpMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkIndex	A far pointer that returns the index of the first mark selected.

Description:

This function returns the index of the mark that is first in the list of selected marks. **ART_mark_selected_next()** returns the index of the next mark.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */

. . .
case ID_EDIT_DELETE:  /*delete all marks   */

ART_mark_selected_first(hIGear, &hMarkIndex);

/* iterate through all marks that were selected by user
and delete them */
while (hMarkIndex != ART_INVALID_ID)
{
hMarkDelete = hMarkIndex;
ART_mark_selected_next(hIGear, &hMarkIndex);
ART_mark_delete(hIGear, hMarkDelete);
}
```

ART_mark_selected_next

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_selected_next(HIGEAR  hIGear,
   LPART_MARK_INDEX lpMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpMarkIndex	A far pointer that returns the index of the first mark selected.

Description:

This function returns the index of the next mark in the list of selected marks. (Or the selected mark following the mark index number currently pointed to). In conjunction with **ART_mark_selected_first()** you can iterate through all of the selected marks.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

See example for **ART_mark_selected_first()** function.

ART_mark_send_to_back

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_send_to_back(HIGEAR hIGear,
   ART_MARK_INDEX hMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier

Description:

This function moves a mark to the back of the stacking order.

NOTE: The mark must be repainted in order to see a change in the stacking order.

An error is set if any of the following conditions are met:

- **hIGear** does not reference a valid ImageGear handle.
- There are no marks with a matching index block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR    hIGear;  /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount;  /* tally of IG errors on  */
      /* stack    */
ART_MARK_INDEX hMarkIndex;  /* index for mark to  */
/* bring to front   */

nErrcount = ART_mark_selected_first(hIGear,&hMarkIndex);
if (hMarkIndex != ART_MARK_INVALID)
{
nErrcount = ART_mark_send_to_back(hIGear,hMarkIndex);
}
```

ART_mark_visible

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_mark_visible(HIGEAR hIGear,
   ART_MARK_INDEX hMarkIndex,
   BOOL fVisible
   );
```

Arguments:

hIGear	HIGEAR handle to the image
hMarkIndex	Mark identifier
fVisible	If set to TRUE, the mark is made visible; FALSE, invisible.

Description:

This function sets the visibility of a mark. When `fVisible` is set to `TRUE`, the function show marks; when `fVisible` is set to `FALSE`, it hides marks. The `bVisible` field of the mark's attribute structure is set to the latest value for `fVisible`.

NOTE: The mark must be repainted in order to see a change in the visibility state.

An error is set if any of the following conditions are met:

- `hIGear` does not reference a valid ImageGear handle.
- There are no marks with a matching index block.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR   hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT  nErrcount; /* tally of IG errors on  */
/* stack    */
ART_MARK_INDEX  hMarkIndex; /* index to current mark */
ART_MARK_ATTRIBUTES ma; /* structures for   */
/* attributes of current  */
/* mark    */

nErrcount = ART_mark_query(hIGear, hMarkIndex, &ma);

nErrcount = ART_mark_is_visible(hIGear, hMarkIndex,
fVisible);
if (fVisible == TRUE && ma.dwType == 10){
/* if an attach-a-note mark is visible  */
nErrcount = ART_mark_visible(hIGear, hMarkIndex,
FALSE);
/* make it invisible    */
}
```

ART_rotation_angle_get

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_rotation_angle_get (HIGEAR IGear,
    LPDWORD lpdwAngle
    );
```

Arguments:

hIGear	HIGEAR handle to the image
lpdwAngle	A pointer to the angle value

Description:

This function determines the angle (in degrees) that an annotation rotates. Values are 0, 90, 180, 270.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of Image  */
DWORD dwAngle;   /* Rotation angle    */
AT_ERRCOUNT nErrCount; /* Tally of IG errors on stack  */

nErrCount = ART_rotation_angle_get (hIGear,&dwAngle);
```

ART_rotation_angle_set

Group: Mark Management

Declaration:

```
AT_ERRCOUNT ART_rotation_angle_set (HIGEAR hIGear,
    DWORD dwAngle
    );
```

Arguments:

hIGear	HIGEAR handle to the image
dwAngle	The angle to rotate

Description:

This function determines the angle (in degrees) that an annotation rotates. Values are 0,90,180,270. The default is 0. Any other value (not 0,90,180,270) is interpreted as 0.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of Image  */
AT_ERRCOUNT nErrCount; /* Tally of IG errors on stack  */

nErrCount = ART_rotation_angle_set (hIGear,90);
```

ART_version_numbers

Declaration:

```
AT_ERRCOUNT ACCUAPI ART_version_numbers(
   DWORD lpdwMajor,
   DWORD lpdwMinor,
   DWORD lpdwUpdate,
   BOOL lpbNewAPI
   );
```

Arguments:

lpdwMajor	Far pointer to an INT variable in which will be stored the Major version number of the version of ART component that you are using
lpdwMinor	Far pointer to an INT variable in which will be stored the Minor version number of the version of ART component that you are using
lpdwUpdate	Far pointer to an INT variable in which will be stored the Update (bug fix) number, reflecting any updates you have received and installed in this version of ART component.
lpbNewAPI	This argument returns TRUE if current version of ART component uses new ImageGear display API.

Description:

This function provides you with information about currently used version of ART component.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

ART Component Macros

ImageGear ART Component provides the following macros:

- **ART_UNIX_LEGACY_FONTS_LOAD_GET**
- **ART_UNIX_LEGACY_FONTS_LOAD_SET**

ART_UNIX_LEGACY_FONTS_LOAD_GET

Group: Macro

Declaration:

```
AT_BOOL ART_UNIX_LEGACY_FONTS_LOAD_GET(LPART_MARK_ATTRIBUTES
pMark);
```

Arguments:

pMark	ART mark.

Description:

This function determines whether the font creation for individual pMark mark is enabled or disabled.

Bits Per Pixel: All

Return Value:

- TRUE - legacy X font creation for individual pMark mark is enabled
- FALSE - legacy X font creation for individual pMark mark is disabled

ART_UNIX_LEGACY_FONTS_LOAD_SET

Group: Macro

Declaration:

```
AT_ERRCOUNT
ART_UNIX_LEGACY_FONTS_LOAD_SET(LPART_MARK_ATTRIBUTES pMark,
AT_BOOL Value)
```

Arguments:

pMark	ART mark.
Value	Boolean parameter: ■ TRUE – enable legacy X font creation for individual pMark mark ■ FALSE – disable legacy X font creation for individual pMark mark

Description:

This function enables or disables legacy X font creation for individual pMark mark.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

ART Component Callback Functions Reference

ImageGear ART Component provides the following callback functions:

- **LPFNIG_ART_CREATENOTIFYFUNC**
- **LPFNIG_ART_DELETENOTIFYFUNC**
- **LPFNIG_ART_DRAWNOTIFYFUNC**
- **LPFNIG_ART_MODIFYNOTIFYFUNC**
- **LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC**
- **LPFNIG_ART_PREMODIFYNOTIFYFUNC**
- **LPFNIG_ART_SELECTNOTIFYFUNC**

LPFNART_GUI_TEXTREPLACEFUNC

Group: Callback

Declaration:

```
typedef BOOL (FAR PASCAL *LPFNART_GUI_TEXTREPLACEFUNC)(
   LPVOID lpPrivate,
   DWORD  dwTextID,
   const LPSTRlpszText,
   LPSTR FAR*lpszNewText
   );
```

Arguments:

lpPrivate	A far pointer to private data being passed to the callback
dwTextID	The button ID
lpszText	The current text string for the tool tip
lpszNewText	The replacement string for the tool tip

Description:

This callback function gets called whenever a "tool tip" is about to be displayed over a button. It gives you a chance to replace the default text for the button's tool tip. Tool tips are the little text boxes that appear when you let the mouse pointer rest over an ART button. For example, if you are over the Filled Rectangle button, a small rectangle appears next to it saying "Filled Rectangle".

NOTE: In order to use this callback, it must be registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Sample: None

LPFNART_GUI_TOOLBARNOTIFYFUNC

Group: Callback

Declaration:

```
typedef BOOL (FAR PASCAL *LPFNART_GUI_TOOLBARNOTIFYFUNC)(
   LPVOIDlpPrivate,
   DWORDdwButtonID,
   UINTnReason
   );
```

Arguments:

lpPrivate	A far pointer to private data being passed to the callback
dwButtonID	The ID for the button that was pressed. These are defined in ARTGUI.H and begin with ART_BUTTON_.
nReason	The reason that the callback is being called. These are defined in ARTGUI.H and include: ART_GUI_TOOLBAR_BUTTON_SELECTED, ART_GUI_TOOLBAR_NONE_SELECTED, ART_GUI_TOOLBAR_HIDDEN, ART_GUI_TOOLBAR_DESTROYED,

Description:

This callback, if registered, gets called when a user presses a button on the toolbar, or when the toolbar is closed via the close menu or Win95's 'x' on it's mini-menu bar.

This callback passes dwButtonID and nReason to let you know what triggered the callback. dwButtonID is the button that was pressed. Each button has a unique ID:

ART_GUI_TOOLBAR_BUTTON_SELECTED	The button dwButtonID was pressed.
ART_GUI_TOOLBAR_NONE_SELECTED	The user presses a button; no other buttons are pressed.
ART_GUI_TOOLBAR_HIDDEN	The user pressed 'x' or closed the toolbar.
ART_GUI_TOOLBAR_DESTROYED	The developer called the Windows API function DestroyWindow() and passed the handle of the toolbar window.

NOTE: In order to use this callback, it must be registered with ART using the **ART_callback_register()** function.

Bits Per Pixel: N/A

Return Value:

The return value is ignored at this time.

Sample: **None**

LPFNIG_ART_CREATENOTIFYFUNC

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI *LPFNART_CREATENOTIFYFUNC)(
   HIGEAR    hIGear,
   LPVOID    lpPrivate,
   ART_MARK_INDEX hMarkIndex,
   LPART_MARK_ATTRIBUTES lpMarkAttr
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpPrivate	A far pointer to the private callback data
hMarkIndex	The index identifier for the mark just created
lpMarkAttr	A far pointer to an ART-defined attribute structure of type AT_MARK_ATTRIBUTES

Description:

This function is called by the library after a mark is created.

If the callback function returns TRUE, the mark is created; if it returns FALSE, the mark is not created. This callback gives you the opportunity to determine whether a mark is created or not.

NOTE: In order to use this callback, it must fbe registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: All

Return Value:

Returns TRUE - mark is created; FALSE - mark isn't created.

Example:

```
BOOL ACCUAPI ARTCreateNotify(HIGEAR hIGear,
LPVOID lpPrivate, ART_MARK_INDEX hMarkIndex,
const LPART_MARK_ATTRIBUTES lpMarkAttr)
{
/* add a named block to each mark that gets created */

ART_mark_block_create(hIGear, hMarkIndex,
myBlockName, myBlockData, myBlockSize);

return TRUE;
}
```

LPFNIG_ART_DELETENOTIFYFUNC

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI *LPFNIG_ART_DELETENOTIFYFUNC)(
   HIGEAR hIGear,
   LPVOID lpPrivate,
   ART_MARK_INDEX hMarkIndex
   );
```

Arguments:

hIGear	HIGEAR handle to the image
lpPrivate	A far pointer to the private callback data
hMarkIndex	The index of the to-be-deleted mark

Description:

This function is called whenever a mark is deleted. If the callback function returns FALSE, the mark is not deleted. It should return TRUE to allow the mark to be deleted.

NOTE: In order to use this callback, it must first be registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: All

Return Value:

If the callback function returns FALSE, the mark is not deleted. It should return TRUE to allow the mark to be deleted.

Example:

```
BOOL ACCUAPI MyDeleteMarkFunc(HIGEAR hIGear, LPVOID
lpPrivate, ART_MARK_INDEX hMarkIndex)
{
/* before deleting the mark, copy it to the clipboard */
ART_edit_copy(hIGear, &hMarkIndex, 1);

return TRUE;
}
```

LPFNIG_ART_DRAWNOTIFYFUNC

Group: Callback

Declaration:

```
typedef VOID (ACCUAPI *LPFNIG_ART_DRAWNOTIFYFUNC)(
   HIGEAR hIGear,
   DWORD dwGrpID,
   LPVOID lpPrivate,
   HDC hDC,
   AT_MODE nWhen,
   ART_MARK_INDEX hMarkIndex,
   const LPART_MARK_ATTRIBUTES lpMarkAttr,
   const LPAT_RECT lprcClip,
   const LPAT_RECT lprcBounds,
   AT_MODE nOrientation
   );
```

Arguments:

hIGear	HIGEAR handle to the image
dwGrpID	Identifier of group
lpPrivate	A far pointer that takes private callback data
hDC	Windows handle to the device context to draw to
nWhen	Before or after image
hMarkIndex	Index for the mark
lpMarkAttr	Set to the ART_MARK_ATTRIBUTES structure of the mark to draw
lprcClip	Set to the AT_RECT coordinates of the clipping rectangle
lprcBounds	The scaled bounds of the mark
nOrientation	The orientation of the mark. Set to an AT_MODE constant (defined in Gear.h) that begins with IG_ORIENT_TOP_ .

Description:

When registered, this callback gets called when ART encounters a mark that it does not know how to draw. The main purpose of this callback is for painting user-defined marks that the developer has created. This callback gets called twice each time the mark needs to be painted. On the first call, the `nWhen` parameter is set to `ART_DISPLAY_PREPAINT`. This indicates that the underlying image has not repainted itself yet. On the second call, `nWhen` is set to `ART_DISPLAY_POSTPAINT`. Any drawing that is done during the pre-paint phase isclobbered by the underlying image when it paints itself. The pre-paint phase is used to create clipping regions for optimized painting. The mark is drawn during the post-paint phase. The `hMarkIndex` parameter indicates that the mark needs to be painted and its attributes are passed in the `lpMarkAttr` parameter. If the mark has extra data in named blocks that is necessary during painting, the named blocks are queried by this function.

The `lprcClip` parameter is the current clipping rectangle. Any drawing that occurs outside this rectangle is clipped by the operating system. This parameter is passed to give the callback an opportunity to optimize it's painting within the rectangle. The `lprcBounds` parameter is the rectangle of the current boundaries of the mark. This parameter differs from the `rcBounds` field in the `ART_MARK_ATTRIBUTES` structure. The rectangle in this parameter is scaled for the display and if the underlying image is rotated, then `lprcBounds` is rotated. The `nOrientation` parameter tells the callback how to orient the mark when drawn. Possible values include: `IG_ROTATE_0`, `IG_ROTATE_90`, `IG_ROTATE_180` and `IG_ROTATE_270`.

NOTE: In order to use this callback, it must be registered with ART using the function **`ART_callback_register()`**.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
VOID ACCUAPI DRAWNOTIFYFUNC(
    HIGEAR hIGear,
    DWORD dwGrpID,
    LPVOID lpPrivate,
    HDC hDC,
    AT_MODE nWhen,
    ART_MARK_INDEX hMarkIndex,
    const LPART_MARK_ATTRIBUTES lpMarkAttr,
    const LPAT_RECT lprcClip,
    const LPAT_RECT lprcBounds,
    AT_MODE nOrientation
    )
{
    HBRUSH hBrush;
    HBRUSH hOldBrush;
    AT_RGBQUAD color = lpMarkAttr->rgbColor1;

    AT_POINT ptPoints[2];

    ptPoints[0].x = lpMarkAttr->rcBounds.left;
    ptPoints[0].y = lpMarkAttr->rcBounds.top;

    ptPoints[1].x = lpMarkAttr->rcBounds.right;
    ptPoints[1].y = lpMarkAttr->rcBounds.bottom;

    IG_dspl_image_to_device(hIGear, dwGrpID, hWnd, hDC,
ptPoints, 2);

    hBrush = CreateSolidBrush(RGB(color.rgbRed, color.rgbGreen ,
color.rgbBlue));
    hOldBrush = SelectObject(hDC, hBrush);
    Ellipse(hDC, ptPoints[0].x, ptPoints[0].y, ptPoints[1].x,
ptPoints[1].y);
    SelectObject(hDC, hOldBrush);
    DeleteObject(hBrush);
}
```

LPFNIG_ART_MODIFYNOTIFYFUNC

Group: Callback

Declaration:

```
typedef VOID (ACCUAPI *LPFNIG_ART_MODIFYNOTIFYFUNC)(
    HIGEAR       hIGear,
    LPVOID       lpPrivate,
```

```
ART_MARK_INDEXhMarkIndex,
const LPART_MARK_ATTRIBUTESlpMarkAttr,
AT_MODE      nReason
);
```

Arguments:

hIGear	HIGEAR handle to the image
lpPrivate	A far pointer to the private callback data
hMarkIndex	The index of the mark just modified
lpMarkAttr	A far pointer to an ART-defined structure of type ART_MARK_ATTRIBUTES
nReason	An integer constant of type AT_MODE, defined in ARTAPI.H. This tells what caused the callback to be called.

Description:

This function is called after a mark has been modified. A mark is modified if it is moved, resized or edited.

Upon entry, nReason is set to a constant indicating what triggered the modification. Your application displays one of the library's standard editing dialog boxes. Here is the list of "reasons" why this callback will be called:

ART_MODIFY_ATTRIBUTES	One or more attributes have been modified (from the ART_MARK_ATTRIBUTES structure)
ART_MODIFY_ANODAT	A named block specific to this type of mark was modified
ART_MODIFY_FILNAM	The filename block was modified
ART_MODIFY_DIB	The embedded DIB was changed
ART_MODIFY_GROUP	The mark's group block was modified
ART_MODIFY_INDEX	The mark's index block was changed
ART_MODIFY_ANTEXT	The orientation or length of text mark have been modified
ART_MODIFY_NAMED_BLOCK	A user-defined named block was modified

NOTE: In order to use this callback, it must be registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
VOID ACCUAPI ARTModifyNotify( HIGEAR hIGear, LPVOID
lpPrivate, ART_MARK_INDEX hMarkIndex,
const LPART_MARK_ATTRIBUTES lpMarkAttr,
AT_MODE nReason)
{
switch (nReason)
{
case ART_MODIFY_ATTRIBUTES:
/* indicate to the user that the highlight */
/* attribute was changed */
    if (lpMarkAttr->bHighlighting)
MessageBox(HWND_DESKTOP, "The mark is “
“highlighted", NULL, MB_OK);
}
}
```

Sample: **None**

LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI *LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC)(
   HIGEAR       hIGear,
   DWORD        dwGrpID,
   LPVOID       lpPrivate,
   HDC          hDC,
   AT_MODE      nWhen,
   ART_MARK_INDEXhMarkIndex,
   const LPART_MARK_ATTRIBUTESlpMarkAttr,
   const LPAT_RECTlprcClip,
   const LPAT_RECTlprcBounds,
   AT_MODE      nOrientation
   );
```

Arguments:

hIGear	HIGEAR handle to the image
dwGrpID	Identifier of group
lpPrivate	A far pointer that takes private callback data
hDC	Windows handle to the drawing device
nWhen	Before or after image
hMarkIndex	Index for the mark
lpMarkAttr	Set to the ART_MARK_ATTRIBUTES structure of the mark to draw
lprcClip	Set to the AT_RECT coordinates of the clipping rectangle
lprcBounds	The scaled bounds of the mark
nOrientation	The orientation of the mark. Set to an AT_MODE constant (defined in Gear.h) that begins with IG_ROTATE_ .

Description:

This callback operates similar to the **LPFNIG_ART_DRAWNOTIFYFUNC** callback. However, this callback is called for known marks, instead of user-defined marks. The purpose of this callback is to let the developer draw the standard marks, instead of letting the library do it.

All of the parameters are the same as LPFNIG_ART_DRAWNOTIFYFUNC. The return value is different. When this callback returns TRUE, it means that the library does not draw the mark. When it returns FALSE, the callback did not handle the drawing of the mark and the library draws the mark. This callback can be used to draw all mark types or just a few marks. The developer can check the dwType field of the lpMarkAttr parameter to determine if they want to draw this type of mark, or let the library handle it.

Finally the nOrientation parameter tells the callback how to orient the mark when drawn. Possible values include IG_ROTATE_0, IG_ROTATE_90, IG_ROTATE_180 and IG_ROTATE_270.

NOTE: In order to use this callback, it must be registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: All

Return Value:

Returns FALSE if the callback did not handle the drawing of the mark and the library did.

Sample: None

LPFNIG_ART_PREMODIFYNOTIFYFUNC

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI *LPFNIG_ART_PREMODIFYNOTIFYFUNC)(HIGEAR
hIGear,
   LPVOID       lpPrivate,
   ART_MARK_INDEXhMarkIndex,
   const PART_MARK_ATTRIBUTESlpMarkAttr,
   AT_MODE      nReason
   );
```

Arguments:

hIGear	HIGEAR handle of an image.
lpPrivate	A far pointer to private callback data.
hMarkIndex	The index number to the mark to be modified.
lpMarkAttr	Set to type ART_MARK_ATTRIBUTES that is associated with the mark.
nReason	An integer constant of type AT_MODE, defined in ARTAPI.H. This indicates what caused the callback.

Description:

This function is called whenever a mark attribute is about to be modified. A mark is modified if it is moved, resized or edited.

If this callback function returns FALSE, the mark does not change. Set it to TRUE to allow the mark to be modified. Upon entry, nReason is set to a constant indicating what triggered the modification. At this point your application can display one of the library's standard editing dialog boxes. "Reasons" this callback is called may be:

ART_MODIFY_ATTRIBUTES	One or more attributes have been modified (from the ART_MARK_ATTRIBUTES structure).
ART_MODIFY_ANODAT	A named block specific to this type of mark was modified.
ART_MODIFY_FILNAM	The filename block was modified.
ART_MODIFY_DIB	The embedded DIB was changed.
ART_MODIFY_GROUP	The mark's group block was modified.
ART_MODIFY_INDEX	The mark's index block was changed.

ART_MODIFY_ANTEXT	The orientation or length of text mark was modified.
ART_MODIFY_NAMED_BLOCK	A user-defined named block was modified.

NOTE: In order to use this callback, it must be registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
BOOL ACCUAPI ARTPreModifyNotify( HIGEAR hIGear,
LPVOID lpPrivate, ART_MARK_INDEX hMarkIndex,
const LPART_MARK_ATTRIBUTES lpMarkAttr,
AT_MODE nReason)
{
switch (nReason)
{
case ART_MODIFY_ATTRIBUTES:
    /* prevent the highlighting attribute from being */
 /* enabled */
 if (lpMarkAttr->bHighlighting)
    return FALSE;
}

return TRUE;
}
```

LPFNIG_ART_SELECTNOTIFYFUNC

Group: Callback

Declaration:

```
typedef BOOL (ACCUAPI *LPFNIG_ART_SELECTNOTIFYFUNC)(HIGEAR
hIGear,
   LPVOID lpPrivate,
   ART_MARK_INDEXhMarkIndex,
   BOOL   fSelect
   );
```

Arguments:

hIGear	HIGEAR handle of an image.
lpPrivate	A far pointer to the private callback data.
hMarkIndex	The index for the selected mark.
fSelect	A flag of type BOOL indicating the new selection state (selected/unselected).

Description:

This function is called by the library when the selection state of a mark is changed. If the callback function returns FALSE, the mark's selection state is not changed. It returns TRUE to allow the selection state to be changed.

NOTE: In order to use this callback, it must be registered with ART using the function **ART_callback_register()**.

Bits Per Pixel: All

Return Value:

Returns FALSE for the mark's selection state to remain unchanged; returns TRUE to allow the mark's selection state to change.

Example:

```
BOOL ACCUAPI ARTSelectNotify(HIGEAR hIGear, LPVOID
lpPrivate, ART_MARK_INDEX hMarkIndex, BOOL fSelect)
{
ART_MARK_ATTRIBUTES  ma;

ART_mark_query(hIGear, hMarkIndex, &ma);

if (ma.dwType == ART_MARK_BUTTON && fSelect == TRUE)
{
    /* prevent button marks from being selected */
    return FALSE;
}

return TRUE;
}
```

LPFNIG_ART_TEXTOVERFLOWNOTIFYFUNC

Group: Callback

Declaration:

```
typedef AT_BOOL (ACCUAPI
*LPFNIG_ART_TEXTOVERFLOWNOTIFYFUNC)(HIGEAR  hIGear,
   LPVOID lpPrivate,
   ART_MARK_INDEXhMarkIndex,
   DWORD  dwMarkType,
   const LPAT_RECTlprcPreferableBounds,
   DWORD  dwTextLength,
   DWORD  dwReserved
   );
```

Arguments:

hIGear	HIGEAR handle of an image.
lpPrivate	A pointer to the private callback data.
hMarkIndex	The index for the selected mark.
dwMarkType	The type of selected mark. Currently it can be ART_MARK_TYPED_TEXT, ART_MARK_ATTACH_A_NOTE, ART_MARK_TEXT_FROM_FILE
lprcPreferableBounds	Preferable mark bounds to draw whole mark text (cut to clip bounds)
dwTextLength	The length of overflowed text (the last NULL is not included). If the text is not overflowed must be set to 0
dwReserved	Reserved for future use. Currently should be set to 0

Description:

This callback function is called after the text mark is resized by mouse dragging and before the mark is repainted. It is called independently on whether the text overflows the mark bounds or not. It is applied for ART_MARK_TYPED_TEXT, ART_MARK_ATTACH_A_NOTE and ART_MARK_TEXT_FROM_FILE marks.

The return value is ignored at this time. Using this callback function customer has possibility to change the mark bounds .

To use this callback, it should be registered using the function:

```
ART_callback_register (hIGear,
ART_CALLBACK_TEXT_OVERFLOW_NOTIFY, TextOverflowNotify, NULL);
```

Bits Per Pixel: All

Return Value:

Returns FALSE for the mark's selection state to remain unchanged; returns TRUE to allow the mark's selection state to change.

Example:

```
HIGEAR hIGear;
HWND hWnd;
ART_MARK_INDEX markIndexes[3]; /* array of processed mark
indexes*/
/* ... */
/* Initialize the markIndexes array anywhere*/
/* and call */
/* ART_callback_register(hIGear,
ART_CALLBACK_TEXT_OVERFLOW_NOTIFY, TextOverflowNotify,
lpPrivate);*/
/* to register TextOverflowNotify callback function*/
/* ... */
AT_BOOL ACCUAPI TextOverflowNotify(
HIGEAR hIGear,
LPVOID lpPrivate,
ART_MARK_INDEX hMarkIndex,
DWORD dwMarkType,
const LPAT_RECT lprcPreferedBounds,
DWORD dwOverflowedTextLength,
DWORD dwReserved
)
{
AT_ERRCOUNT nErrCount;
ART_MARK_ATTRIBUTES ma;
AT_RECT rcBounds;
DWORD dwTextLength;
/* The ART_GUI_text_flow is applied only for ART_MARK_TYPED_TEXT
and ART_MARK_ATTACH_A_NOTE */
if ( dwMarkType != ART_MARK_TYPED_TEXT && dwMarkType !=
ART_MARK_ATTACH_A_NOTE )
return TRUE;
if ( hMarkIndex == markIndexes[0] )
{
nErrCount = ART_GUI_text_flow(hIGear, IG_GRP_DEFAULT, hWnd,
(LPART_MARK_INDEX)markIndexes, 3 );
if ( nErrCount == 0 )
nErrCount = ART_GUI_mark_paint( hIGear, IG_GRP_DEFAULT,
markIndexes[1], hWnd, TRUE );
}
else if ( hMarkIndex == markIndexes[1] )
{
nErrCount = ART_GUI_text_flow( hIGear, 0, hWnd,
(LPART_MARK_INDEX)(markIndexes + 1), 2 );
}
```

```
/* Set preferable bounds to the third mark */
/* If you do not want to change the mark size remove the
following code */
if ( hMarkIndex != markIndexes[2] && nErrCount == 0 )
{
ART_GUI_mark_paint( hIGear, IG_GRP_DEFAULT, markIndexes[2],
hWnd,FALSE );
nErrCount = ART_mark_query( hIGear, markIndexes[2], &ma );
/* Get the preferable bounds for third mark*/
if ( nErrCount == 0 )
nErrCount = ART_GUI_text_overflow_check( hIGear,
IG_GRP_DEFAULT,
hWnd, markIndexes[2], &rcBounds, &dwTextLength);
if ( nErrCount == 0 )
{
/*Change the bounds*/
ma.rcBounds = rcBounds;
ART_mark_modify( hIGear, markIndexes[2], &ma );
/*Repaint the mark*/
ART_GUI_mark_paint( hIGear, IG_GRP_DEFAULT, markIndexes[2],
hWnd, TRUE );
}
}
return TRUE;
}
```

Sample: **ART**

MD Component API Function Reference

The ImageGear MD Component is available on Linux platforms only.

You can call MD component functions using two ways. Firstly, you may call this function through it macro defined in **i_MED.h** public header file:

```
#define MED_DCM_load_DICOM(_lpFileName, _lphIGear, _nSyntax,
_page_number)\ (AT_ERRCOUNT (CACCUAPI *)(const LPSTR, \const
LPCHAR, LPHIGEAR, const AT_MODE, const UINT)) \
     IG_comm_function_call)("MED.MED_DCM_load_DICOM", \
(_lpFileName),(_lphIGear),(_nSyntax), (_page_number))
```

In this case your application will search all included public headers for this macro and call this function through **IG_comm_function_call()** component manager function call that is also determined in **i_MED.h**. So if you are going to use MD component function in a multiple loop we would recommend you the second way of using this function type declaration that is also determined in **i_MED.h** file:

```
typedef  AT_ERRCOUNT (LPACCUAPI LPAFT_MED_DCM_LOAD_DICOM)(
const LPSTRlpszFileName,
LPHIGEAR  lphIGear,
const AT_MODEnSyntax,
const UINTpage_number
 );
```

So you should declare the variable of this function type and then use **IG_comm_entry_request()** function to initialize this variable with correct value and call it then.

MED_DCM_DS_bits_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_bits_get(
const HIGEAR hIGear,
LPUINT lpBitsAllocated,
LPUINT lpBitsStored,
LPUINT lpHighBit,
LPUINT lpSamplesPerPix);
```

Arguments:

hIGear	**HIGEAR** handle of the image containing a DICOM Data Set
lpBitsAllocated	A far pointer that returns the number of Bits Allocated (0028,0100) for each pixel in the image Set this to **NULL** if you do not need this information
lpBitsStored	A far pointer that returns the number of Bits Stored (0028,0101) for each pixel in the image. Set this to **NULL** if you do not need this information.
lpHighBit	A far pointer that returns the position of the High Bit (0028,0102) of the pixels in a DICOM image. Set this to **NULL** if you do not need this information.
lpSamplePerPix	A far pointer that returns the number of Samples Per Pixel (0028,0002) for the DICOM image. Set this to **NULL** if you do not need this information.

Description:

This function returns three critical DE values from the Data Set: the number of Bits Allocated (0028,0100), the number of Bits Stored (0028,0101), and the position of the High Bit (0028,0102). These values are stored in the HDS structure pointed to by the **HIGEAR**.

The value returned by **lpBitsAllocated** is not always the same as the number of bits per pixel for the image's DIB. It is the number of bits allocated per sample for each pixel. A 24-bit RGB image would return a Bits Allocated of 8 since each of the 3 samples has 8 bits allocated for it.

The Bits Stored is the number of bits actually used out of the total available (Bits Allocated). You can have a 16-bit gray-scale image that only actually uses 12-bits. In this case, the Bits Allocated would be 16 and Bits Stored would be 12. The Bits Stored is always less than or equal to the Bits Allocated.

The High Bit shows where the Bits Stored are placed in the Bits Allocated **WORD** or **DWORD**. Since the Bits Stored can be less than the Bits Allocated, the Bits Stored can be placed in the Bits Allocated with different starting points. This value tells you where the Bits Stored actually resides. High Bits is always less than Bits Allocated.

See also:

- **MED_IP_high_bit_transform()**
- **MED_IP_reduce_depth_with_LUT()**
- **MED_IP_reduce_depth_with_downshift()**

Bits Per Pixel: All

Return Value:

Returns the number of ImageGear errors that occurred during the function call

MED_DCM_DS_create

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_create(
HIGEAR hIGear,
const AT_MODE Transfer_syntax);
```

Arguments:

hIGear	**HIGEAR** handle to the image
Transfer_syntax	Set this variable to the desired Transfer Syntax (TS) with which to create the Data Set. Use one of the ImageGear defined constants defined in DCM.h such as: **MED_DCM_TS_IMPLICIT_VR_LE**, **MED_DCM_TS_EXPLICIT_VR_LE,** **MED_DCM_TS_EXPLICIT_VR_BE**

Description:

This function creates a Data Set. If there is already a Data Set that is associated with the **HIGEAR** image, it will be replaced. In addition, the absolute minimum of Critical DEs (Data Elements), such as Pixel Representation and Samples per Pixel will be automatically added to the Data Set.

Below is a list of those Mandatory DEs that will be filled out automatically by this function. See ***Part 6:*** *Data Dictionary of the DICOM Specification for the definitions of these DEs*.

Note that there are two different Group Numbers listed below, and that the Tag numbers (the second numbers shown in the parentheses) identify which Data Element will be filled in:

```
Affected DEs from Tag.group= 0x0028:
(0028,0010)/* Rows*/
(0028,0011)/* Columns*/
(0028,0100)/* Bits Allocated*/
(0028,0101)/* Bits Stored*/
(0028,0102)/* High Bit*/
(0028,0004)/* Photometric Interpretation*/
(0028,0103)/* Pixel Representation*/
(0028,0002)/* Samples per pixel*/
(0028,0006)/* Planar Configuration*/
Affected DEs from Tag.group= 0x7FE0;
(7FE0,0010)/* Pixel Data*/
```

See Working With DICOM Data Structures section for more information.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR       hIGear;
AT_MODE      Transfer_syntax;

IG_load_file("image1.tif", &hIGear);
Transfer_syntax = MED_DCM_TS_IMPLICIT_VR_LE;
MED_DCM_DS_create(hIGear, Transfer_syntax);
```

MED_DCM_DS_curr_data_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_data_get(
const HIGEAR hIGear,
LPVOID lapdog,
const DWORD size_of_data,
LPAT_DCM_VL lpActualSize);
```

Arguments:

hIGear	**HIGEAR** handle to the image to get data from
lpData	A far pointer to a **VOID** buffer into which the data will be copied.
size_of_lpData	Size of above buffer, **lpData**
lpActualSize	Actual number of bytes copied to **lpData** (**NULL** if do not care). This will always be equal to or smaller than **size_of_lpData**. If **lpActualSize** is less than the VL of the current Data Element then **lpData** does not contain all the data because **lpData** does not point to enough memory to hold the entire object.

Description:

This function returns the data (Value Field) of the Current Data Element in its native form. Each Data Element can be of one of many different data types (int, word, bytes, float, double, string, etc.) You must know the VR (Value Representation) of the data in order to use this data. To get the VR and the VL (Value Length), you can use one of the **MED_DCM_DS_move_…()** functions or **MED_DCM_DS_curr_info_get()**. Each time you move the Current Data Element with a **_move_…()** function, the VR and VL of the new Current Data Element are returned to you.

To retrieve the data as a string use **MED_DCM_DS_curr_data_get_string()**.

NOTES: Some VRs depend on the byte order of your operating system (Big Endian or Little Endian) and the Transfer Syntax of the DICOM file. Examples of such VRs are: **WORD**, **LONG**, **FLOAT**, **DOUBLE**, etc. The Data Field values will be returned to you already adjusted to the proper format and no byte-swapping is needed.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_curr_data_get_string

Group: Data Set Utility

Declaration:

```
BOOL ACCUAPI MED_DCM_DS_curr_data_get_string (const HIGEAR
hIGear,
             LPCHAR lpString,
             const DWORD size_of_lpstring);
```

Arguments:

hIGear	**HIGEAR** handle to the image to get data from
lpString	A far pointer to a memory location which will be filled with the data from the Current Data Element as a **NULL**-terminated character string.
size_of_lpString	The variable which tells the function the length of **lpString**, in bytes, expressed as a **DWORD.** If you specify a string length that is not long enough to hold the data, the data will simply be truncated.

Description:

This function gets the data from the Current Data Element, and always returns it to you as a **NULL**-terminated character string. The marker to the Current Data Element can be moved using one of the **MED_DCM_DS_move_…()** functions.

Data Elements can be of many different data types (int, word, bytes, float, double, string, etc.), but this function will always convert the data to a string. To return the data in its “natural form”, use **MED_DCM_DS_curr_data_get()**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_curr_data_set

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_data_set(HIGEAR hIGear,
                    const LPVOID lpData,
                    const DWORD size_of_data);
```

Arguments:

hIGear	**HIGEAR** handle of the image
lpData	A far pointer of type **VOID**. Set this to the data you would like stored into the Data Field of the Current Data Element
size_of_data	Set this **DWORD** variable to the size of the data in **lpData**.

Description:

This function allows you to overwrite the Value Field (data) of the Current Data Element, by copying the data from your buffer to the HDS table. The Data must be of the correct data type to match the Value Representation (VR) of the Current DE. You can use **MED_DCM_DS_curr_info_get()** to find out the VR of the Current DE, and you can use one of the **MED_DCM_DS_move_…()** functions to set the Current Data Element.

If the Data Value can accept multiple Data Elements, the data values should be set as a single block of memory. To query the Value Multiplicity (VM) use **MED_DCM_util_tag_info_get()**.

The length of a DICOM Data Field must always be an even number. If you set the **size_of_lpData** to an odd number of bytes, ImageGear will pad it (and your data) to make it an even-numbered length.

An error is set if the data type does not match the VR of the Current DE.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_curr_index_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_index_get(const HIGEAR
hIGear,
                    LPDWORD lpIndex);
```

Arguments:

hIGearc	The **HIGEAR** handle to the image being queried
lpIndex	A far pointer of type **DWORD** which returns with the index of the Current Data Element

Description:

Returns the index of the Current Data Element in the Data Set associated with the current **HIGEAR** image. This function indexes through all levels of the Data Set. The index of the first Data Element in the table is 0.

To move to a specific index and make it the Current Data Element, use **MED_DCM_DS_move_index().**

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_curr_info_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_info_get(const HIGEAR
hIGear,
                    LPAT_DCM_TAG lpTag,
                    LPAT_DCM_VR lpVR,
                    LPAT_DCM_VL lpVL,
                    LPWORD lpLevel,
                    LPDWORD lpItem_count);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**
lpVR	A far pointer to variable of type **AT_DCM_VR** which returns the Value Representation (VR) of the Current Data Element; set this to **NULL** if you do not need this information.
lpVL	A far pointer to a variable of type **DWORD** that returns the length, in bytes, of the Current Data Element's Data Field; set this to **NULL** if you do not need this information.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL** if you do not need this information.
lpItem_count	A far pointer that returns the number of items stored in the Data Field; set this to **NULL** if you do not need this information.

Description:

This function returns information about the Current Data Element (DE). It operates in just the same way as the **MED_DCM_move_…()** functions except that it does not change the Current DE - it only reports about it.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_curr_remove

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_remove (HIGEAR hIGear,
                    LPBOOL lpRemoved);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpRemoved	A far pointer to a **BOOL** which returns **TRUE** if the Current Data Element was removed; it will return **FALSE** if there are no more removable DEs in the Data Set or if the Current Data Element could not be removed.

Description:

This function removes the Current Data Element from the Data Set. The new Current Data Element will be the Data Element following the one that was just removed, unless you have deleted the last DE, in which case the CDE will be the "new" last DE.

If the Data Set is empty (there are no removable DEs remaining) **lpRemoved** will return **FALSE**. The critical DEs cannot be removed. Non-removable DEs also include Sequence and Item Delimiters.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_DE_insert

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_DE_insert(HIGEAR hIGear,
                     const AT_DCM_TAG Tag,
                     const AT_DCM_VR vr,
                     const LPVOID lpData,
                     const DWORD size_of_data);
```

Arguments:

hIGear	**HIGEAR** handle to the image
Tag	Set to a Tag value. The Tag must be supplied as a 32-bit value in which the first 16 bits (WORD) represent the Group Number and the second 16 bits represent the Element Number. (See **DCM.h**).
vr	Set to the VR (Value Representation) of the Data Element to be inserted. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpData	A far **VOID** pointer to the data that you would like to insert.
size_of_data	Set this **DWORD** variable to the size of the data in **lpData**.

Description:

This function inserts a Data Element into the Data Set. Your new Data Element will be placed into the Data Set sorted by its Tag value on the same level as that of the Current Data Element. If the DE already exists, the new one overwrites it. Specifying a Group Length DE does not cause an error, but simply is ignored. Your data will be padded to an even length if necessary.

NOTE: As of this writing, there are no constants defined for those Data Elements that have a VR of “CS.” Refer to Part 3 of the Specification for the valid Code Strings which you can enter for data of type CS (Code String). Note also that the length of a Code String is the number of characters between the parentheses.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;
MED_DCM_DS_DE_insert(hIGear, DCM_TAG_PhotometricInterpretation,
MED_DCM_VR_CS, “MONOCHROME2”, 11);
```

MED_DCM_DS_destroy

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_destroy(HIGEAR hIGear);
```

Arguments:

hIGear	**HIGEAR** handle of the image whose Data Set will be destroyed.

Description:

This function destroys the Data Set associated with the **HIGEAR** image specified. Once the Data Set is removed, the image is just like any other image loaded into ImageGear.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_exists

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_exists(const HIGEAR hIGear,
                    LPBOOL lpExists);
```

Arguments:

hIGear	A **HIGEAR** handle to an image
lpExists	A far pointer that returns a **BOOL**. If it returns **TRUE**, a Data Set exists for the image; if it returns **FALSE**, a Data Set does not exist for the image.

Description:

This function determines whether an image has a Data Set associated with it. Supply it with the **HIGEAR** handle to the image you want to check, and **lpExists** will return whether or not it has a Data Set.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_info_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_info_get(const HIGEAR hIGear,
                    LPDWORD lpNumTags,
                    LPDWORD lpMaxLevel);
```

Arguments:

hIGear	**HIGEAR** handle to an image
lpNumTags	A far pointer to a **DWORD** which returns the number of Data Elements (same as the number of Tags) associated with the image's Data Set.
lpMaxLevel	A far pointer to a **DWORD** which returns the maximum SQ Level of the Data Set.

Description:

This function returns the number of Data Elements (DEs) associated with the Data Set of the **HIGEAR** image. You might use the number of DEs returned to set the limit on a loop that iterates through each Data Element in a Data Set. If there are no SQ Data Elements, **lpMaxLevel** is set to 0.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_is_empty

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_is_empty(const HIGEAR hIGear,
                    LPBOOL lpIsEmpty);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpIsEmpty	A far pointer to a **BOOL** which returns the status of the Tags in the Data Set

Description:

This function returns a **TRUE** through **lpIsEmpty** argument if the Data Set associated with **HIGEAR** has no Data Elements in it.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_LUT_copy_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_LUT_copy_get(
   HIGEAR hIGear,
   HIGEAR hIGearPresstate,
   AT_DCM_TAG lutSqTag,
   HIGLUT *lpLUT
);
```

Arguments:

hIGear	Image, which dataset is checked for presence of the LUT.
hIGearPresstate	Presentation state HIGEAR, which dataset is checked for presence of the LUT. Set to NULL if no presentation state HIGEAR is available.
lutSqTag	Specifies the LUT sequence.
lpLUT	Returns new HIGLUT object with the LUT obtained from a DataSet.

Description:

This function obtains a new copy of a specified LUT from either presentation state HIGEAR (hIGearPresState), or the image HIGEAR (hIGear). If the LUT exists in both the Presentation state HIGEAR, and in the image HIGEAR, the function returns the LUT from Presentation State HIGEAR.

Use **IG_LUT_destroy()** to destroy the LUT returned from this function, when it is no longer neded.

This function supports the following LUT sequences:

- DCM_TAG_ModalityLUTSequence
- DCM_TAG_VOILUTSequence
- DCM_TAG_PresentationLUTSequence

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_LUT_exists

Group: Data Set Utility

Declaration:

```
AT_BOOL ACCUAPI MED_DCM_DS_LUT_exists(
    HIGEAR hIGear,
    HIGEAR hIGearPresstate,
    AT_DCM_TAG lutSqTag
);
```

Arguments:

hIGear	Image, which dataset is checked for presence of the LUT
hIGearPresstate	Presentation state HIGEAR, which dataset is checked for presence of the LUT. Set to NULL if no presentation state HIGEAR is available.
lutSqTag	Specifies the LUT sequence .

Description:

This function checks whether a presentation state HIGEAR (hIGearPresState), or the image HIGEAR (hIGear) contain specified LUT sequence.

This function supports the following LUT sequences:

- DCM_TAG_ModalityLUTSequence
- DCM_TAG_VOILUTSequence
- DCM_TAG_PresentationLUTSequence

Bits Per Pixel: 8...16-bit

Return Value:

`TRUE` if the DataSet attached to the `HIGEAR` has a specified LUT sequence, `FALSE` - otherwise.

MED_DCM_DS_LUT_update_from

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_LUT_update_from(
    HIGEAR hIGear,
    AT_DCM_TAG lutSqTag,
    HIGLUT lut
);
```

Arguments:

hIGear	**`HIGEAR`** handle to the image
lutSqTag	LUT sequence.
lut	LUT handle.

Description:

This function adds specified LUT to the DataSet.

The function supports the following LUTs:

LUT	Sequence Tag
Modality LUT	DCM_TAG_ModalityLUTSequence
VOI LUT	DCM_TAG_VOILUTSequence, DCM_TAG_SoftcopyVOILUTSequence
Presentation LUT	DCM_TAG_PresentationLUTSequence

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_ascend

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_ascend(HIGEAR hIGear,
   LPAT_DCM_TAG  lpTag,
   LPAT_DCM_VR   lpVR,
   LPAT_DCM_VL   lpVL,
   LPWORD        lpLevel,
   LPDWORD       lpICount);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**.
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a type defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.

Description:

This function moves the Current Data Element up one level. This is only applicable if the Current Data Element is positioned within an SQ Data Element or within an Item Data Element.

If the Current Data Element is positioned anywhere in an SQ, the Current Data Element ascends to the SQ. If within an Item, the new Current DE becomes the Item DE. Only one level is ascended per call.

If the Current DE is at the top level (0), no action is taken.

If this function is successful the **lpLevel** decreases in value by 1. Zero refers to the top level. As the number gets larger the Data Element is deeper into the hierarchy. Data Elements (as well as SQs and Item Delimiters) are always stored in even-numbered levels; odd-numbered levels contain Items and SQ Delimiters.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_descend

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_descend (
const HIGEAR hIGear,
LPAT_DCM_TAG lpTag,
LPAT_DCM_VR  lpVR,
LPAT_DCM_VL lpVL,
LPWORD       lpLevel,
LPDWORD      lpICount);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**.
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.

Description:

This function moves the Current Data Element down one level. This is only applicable if the Current Data Element is positioned at an SQ Data Element or at an Item Data Element.

If the Current Data Element is positioned at an SQ Data Element, the Current Data Element will descend to the first Item in the sequence. If at an Item, the new Current DE becomes the first DE within the Item. Only one level is descended per call.

If the Current DE does not point to an SQ or Item DE, no action is taken and **lpLevel** returns the Current Level.

If this function is successful the **lpLevel** increments from its Current DE level. Zero refers to the top level. As the number gets larger the Data Element is deeper into the hierarchy. Data Elements (as well as SQs and Item Delimiters) are always stored in odd-numbered levels; even-numbered levels contain Items and SQ Delimiters.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_find

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_find(HIGEAR hIGear,
   const AT_MODE    level_op,
   const AT_DCM_TAG Tag,
   LPAT_DCM_VR      lpVR,
   LPAT_DCM_VL    lpVL,
   LPDWORD          lpLevel,
   LPDWORD          lpICount,
   LPBOOL           lpTagFound);
```

Arguments:

hIGear	**HIGEAR** handle to the image
level_op	A variable of type **AT_MODE** that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: **MED_DCM_MOVE_LEVEL_FIXED:** This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ you can only move about the SQ. **MED_DCM_MOVE_LEVEL_FLOAT**: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
Tag	Set to a value of type **DWORD** that identifies the Tag value you would like to search for. The first 16 bits of the **DWORD** represent the Group Number; the second 16 bits represent the Element Number.
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.
lpTagFound	Returns **TRUE** is the Tag was found; **FALSE** otherwise.

Description:

This function searches the Data Set associated with **hIGear** for the Tag specified in **lpTag**. If the Tag is found, the Data Element becomes the Current Data Element. This function also returns the VR (Value Representation), the number of bytes in the Tag's data, and the Item Count.

The levels of the Data Set that will be searched depends on the setting of **level_op**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_find_first

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_find_first(const HIGEAR
hIGear,
   const AT_MODE level_op,
   const WORD    GroupNum,
   LPAT_DCM_VR   lpVR,
   LPAT_DCM_VL lpVL,
   LPDWORD       lpLevel,
   LPDWORD       lpItem_count,
   LPBOOL        lpTagFound);
```

Arguments:

hIGear	**HIGEAR** handle of the image
level_op	A variable of type **AT_MODE** that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: **MED_DCM_MOVE_LEVEL_FIXED:** This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ you can only move about the SQ. **MED_DCM_MOVE_LEVEL_FLOAT**: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
GroupNum	Set this **WORD** variable to the Group Number to search for
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.
lpTagFound	Returns **TRUE** is the Tag was found; **FALSE** otherwise.

Description:

This function searches the Data Set associated with the **HIGEAR** image for the first Tag of the Group Number specified by **GroupNum**. When the first Tag with group **GroupNum** is found, the Data Element is made the Current Data Element. This function also returns the VR (Value Representation), the number of bytes in the Tag's data, and the Item Count.

Which levels of the Data Set will be considered depends on the setting of **level_op**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_first

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_first(const HIGEAR hIGear,
   const AT_MODE level_op,
   LPAT_DCM_TAG  lpTag,
   LPAT_DCM_VR   lpVR,
   LPAT_DCM_VL lpVL,
   LPWORD        lpLevel,
   LPDWORD       lpICount);
```

Arguments:

hIGear	**HIGEAR** handle to the image
level_op	A variable of type **AT_MODE** that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: **MED_DCM_MOVE_LEVEL_FIXED:** This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ you can only move about the SQ. **MED_DCM_MOVE_LEVEL_FLOAT**: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**.
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.

Description:

This function makes the first Data Element *in the specified level* the Current Data Element. This function also returns the DE's VR (Value Representation), the number of bytes in the Tag's data, and the Item Count.

Which level of the Data Set will be considered depends upon the setting of **level_op**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_index

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_index(const HIGEAR hIGear,
   const DWORD  index,
   LPAT_DCM_TAG lpTag,
   LPAT_DCM_VR  lpVR,
   LPAT_DCM_VLlpVL,
   LPDWORD      lpLevel,
   LPDWORD      lpICount);
```

Arguments:

hIGear	The **HIGEAR** handle to the image
index	Set to a **DWORD** that indicates the index that you would like to become the new Current Data Element.
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**.
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.

Description:

This function moves the Current Data Element to the Data Element in the Data Set indicated by **index,** which represent a particular index in the array of Data Elements that make up the Data Set. This function also returns the Tag's VR (Value Representation), its Tag value, the number of bytes in the Tag's data, and the Item Count. The index refers to which element in the Data Element array should be used. The first Data Element will always have an index of 0, and the last Data Element will always have an index of (Total # of DEs - 1). Use **MED_DCM_DS_info_get()** to find out the total # of Data Elements associated with an image's Data Set.

This function pays no attention to Levels. It only returns you the level of the Current Data Element.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_last

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_last(const HIGEAR hIGear,
   const AT_MODE level_op,
   LPAT_DCM_TAG  lpTag,
   LPAT_DCM_VR   lpVR,
   LPAT_DCM_VL lpVL,
   LPWORD        lpLevel,
   LPDWORD       lpICount);
```

Arguments:

hIGear	**HIGEAR** handle to the image
level_op	A variable of type **AT_MODE** that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: **MED_DCM_MOVE_LEVEL_FIXED:** This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ you can only move about the SQ. **MED_DCM_MOVE_LEVEL_FLOAT**: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.

Description:

This function moves the Current Data Element to the last Data Element in the Data Set, or in the current level, depending on **level_op** setting.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_move_next

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_next(const HIGEAR hIGear,
   const AT_MODE level_op,
   LPAT_DCM_TAG  lpTag;
   LPAT_DCM_VR   lpVR,
   LPAT_DCM_VL lpVL,
   LPWORD        lpLevel,
   LPDWORD       lpICount,
   LPLONG        lpNumRemaining);
```

Arguments:

hIGear	**HIGEAR** handle to the image
level_op	A variable of type **AT_MODE** that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: **MED_DCM_MOVE_LEVEL_FIXED:** This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ you can only move about the SQ. **MED_DCM_MOVE_LEVEL_FLOAT**: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.
lpNumRemaining	A far pointer to a **LONG** that returns the number of DE remaining until the end is reached. If the returned value is 0, you are now at last DE, if the returned value is -1, you have attempted to move past the last DE.

Description:

This function moves the Current Data Element to the next Data Element in the Data Set. The value of **lpNumRemaining** tells you whether the Tag has now become the last Tag in the list.

The **level_op** setting determines whether the Current Data Element can move from level to level, or has to stay on the same level.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
        AT_ERRCOUNT nErrcount;
        HIGEAR    hIGear;
        BOOL      IsLast;
        AT_DCM_TAG lpTag;
        AT_DCM_VR lpVR;
        AT_DCM_VL lpVL;
        DWORD     lpICount;
        DWORD     size_of_lpData;

        IsLast = FALSE:
          MED_DCM_DS_move_first(hIGear, &lpTag, &lpVR, &lpVL,
        &lpICount);

        /* iterate through all Data Elements returning the Tag, VR, VL
        and item count of each one. End the loop when lpIsLast == TRUE
        */

        while (IsLast == FALSE) {
           MED_DCM_DS_curr_data_get(hIGear, lpData, size_of_lpData);
           MED_DCM_DS_move_next(hIGear, &lpTag, &lpVR, &lpVL,
        &lpICount, &lpIsLast);
}
```

MED_DCM_DS_move_prev

Group: Data Set Navigation

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_prev(const HIGEAR hIGear,
   const AT_MODE level_op,
   LPAT_DCM_TAG  lpTag;
   LPAT_DCM_VR   lpVR,
   LPAT_DCM_VL lpVL,
   LPWORD        lpLevel,
   LPDWORD       lpICount,
   LPLONG        lpNumRemaining);
```

Arguments:

hIGear	**HIGEAR** handle to the image
level_op	A variable of type **AT_MODE** that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: **MED_DCM_MOVE_LEVEL_FIXED:** This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ you can only move about the SQ. **MED_DCM_MOVE_LEVEL_FLOAT**: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ the Current DE moves back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	A far pointer that returns a 32-bit value of type **AT_DCM_TAG** indicating the numerical value of the Tag of the Current Data Element; set this to **NULL** if you do not need this information. The numerical values of the DICOM Tags are defined in **DataDict.h**
lpVR	A far pointer which returns the new current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVL	A far pointer which returns the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.
lpLevel	A far pointer to a **WORD** which returns the level in the hierarchy of the new Current Data Element; set to **NULL**, if you do not need this information.
lpICount	Returns the Item Count of the data; set to **NULL** if you don't need this information.
lpNumRemaining	A far pointer to a **LONG** that returns the number of DE remaining until the top is reached. If the returned value is 0, you are now at first DE, if the returned value is -1, you have attempted to move past the first DE.

Description:

This function moves the Current Data Element to the previous Data Element in the Data Set. The value of **lpNumRemaining** tells you whether the Tag has now become the first Tag in the list.

The **level_op** setting determines whether the Current Data Element can move from level to level, or has to stay on the same level.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_orig_TS_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_orig_TS_get(const HIGEAR hIGear,
   LPAT_MODE lpOrigTS,
   LPBOOL    lpPart10,
   LPBOOL    lpGrpLengths);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpOrigTS	A far pointer to the original setting for TS (Transfer Syntax). Set to **NULL** if you do not need this information. Transfer Syntax constants are defined in **DataDict.h** as: **MED_DCM_TS_IMPLICIT_VR_LE**, **MED_DCM_TS_EXPLICIT_VR_LE**, etc.
lpPart10	A far pointer to a **BOOL** which indicates whether the original file was a Part 10 file. If **TRUE**–the original was a Part 10 file; If **FALSE**–the original was *not* a Part 10 file. Set to **NULL** if you do not need this information.
lpGrpLengths	A far pointer that returns a Boolean value indicating whether the original DICOM image had any Group Length Data Elements in it. **TRUE** means that it did have Group Length Data Elements; **FALSE** means that there were none. Set to **NULL** if you do not need this information.

Description:

This function returns the original Transfer Syntax used for the image, and indicates whether of not it had a Part 10 header and the group length Data Elements.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_part10_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_part10_get(const HIGEAR hIGear,
   const AT_MODE part10_item,
   const DWORD size_of_lpData,
   LPVOID       lpData,
   LPDWORD      lpSize_of_item);
```

Arguments:

hIGear	**HIGEAR** handle to the image
part10_item	Set to the item in Part 10 that you would like to get. Use one of the **AT_MODE** constants defined in DCM.h that begin with **DCM_PART10_ITEM_SET_.**
size_of_lpData	Set to the size of the buffer **lpData** that you allocate to receive the data. This function will not copy more than this amount of data from the Part 10 Header, stored in the **HIGEAR**, to this buffer.
lpData	A far pointer to a **VOID** buffer in which return the data from the item specified by **part10_item**.
lpSize_of_Item	A far pointer that returns the actual size of the Data Value that is stored in the Part 10 Header. It is not always the size of the data being returned in **lpData**. If this parameter returns a value greater than **size_of_lpData** then **lpData** does not contain all of the data–some has been clipped.

Description:

This function returns the data from the item in the Part 10 header identified by **part10_item**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
#define BUFF_SIZE150
HIGEAR    hIGear;
char          data[BUFF_SIZE];

MED_DCM_DS_part10_get(g_hIGear, DCM_PART10_ITEM_PREAMBLE,
BUFF_SIZE, data, &size_of_data);
```

MED_DCM_DS_part10_set

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_part10_set(const HIGEAR hIGear,
   const AT_MODE   part10_item,
   const LPVOID    lpData,
   const AT_DCM_VL  vl);
```

Arguments:

hIGear	**HIGEAR** handle to the image
part10_item	Set this variable to the type of Part 10 item you would like to set. Use one of the constants defined in **DCM.h** that begin with **DCM_PART10_ITEM_**
lpData	A far **VOID** pointer to the data that you would like to store into the Part 10 item.
vl	Set to the length of the Data Field, in bytes. **AT_DCM_VL** is a data type defined in **DCM.h**.

Description:

This function sets the data of the item in the Part 10 header identified by **part10_item**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_PixPadVal_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_PixPadVal_get(HIGEAR hIGear,
   LPBOOL lpUse_Pix_Padding,
   LPLONG lpPix_Padding_Val,
   LPBYTE lpShow_PPV_as);
```

Arguments:

hIGear	**HIGEAR** handle to the image for which you would like to set the value of the “Pixel Padding Value” Data Element (0028,0120).
lpUse_Pix_Padding	Returns **TRUE** if the value of Pixel Padding Value (0028,0120) will be used; **FALSE** if the value of Pixel Padding Value will be ignored.
lpPix_Padding_Val	Returns the grayscale value that will be used for padding. This value read is that which is stored in the Pixel Padding Value Data Element (0028,0120) of the internal Data Set.
lpShow_PPV_as	Returns the gray-scale value that will be used to display pixels equal to the Pixel Padding Value.

Description:

This function retrieves the Pixel Padding Value (PPV) that is being used for the display of 16-bit grayscale images. This function also returns whether or not the PPV value will be used. The Pixel Padding Value is most often used to fill in the regions around a circular image. This function does ***not*** retrieve the PPV Data Element from the Data Set attached to the **HIGEAR**. It gets the value for PPV from the Internal Data Set (HDS) (which may be equal to the value in the actual Data Set). The purpose of this function is to let you know whether Pixel Padding is set on and off, what, if any value it has, and what color it is set to display as.

Please see the description for **MED_DCM_DS_PixPadVal_set()** for a complete description of how ImageGear handles the Pixel Padding Value Data Element.

To turn off the Pixel Padding Value or to alter the value being used use **MED_DCM_DS_PixPadVal_set()**.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
LONG   Pix_Padding_Val;
BYTE   Show_PPV_as;

nErrcount = MED_DCM_DS_PixPadVal_get(g_hIGear, NULL,
&Pix_Padding_Val, &Show_PPV_as);
```

MED_DCM_DS_PixPadVal_set

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_PixPadVal_set(HIGEAR hIGear,
   const BOOL Use_Pix_Padding,
   const LONG Pix_Padding_Val,
   const BYTE Show_PPV_as);
```

Arguments:

hIGear	**HIGEAR** handle to the image for which you would like to set the value of the internal representation of the “Pixel Padding ValueData Element (0028,0120)”.
Use_Pix_Padding	Set this to **TRUE** to use the value of Pixel Padding Value (0028,0120); **FALSE** to ignore the value of Pixel Padding Value.
Pix_Padding_Val	Set this argument to the gray-scale value to use for the image. This value will be stored in the Pixel Padding Value Data Element (0028,0120) of the internal Data Set. This new value is the value that will be locked into the 16x8 LUT when Pixel Padding is turned on.
Show_PPV_as	Set to the gray-scale value that will be used to display pixels equal to the Pixel Padding Value.

Description:

This function is used to set the Pixel Padding Value that is to be used while displaying a 16-bit grayscale image. Here is a description of what this Data Element is used for and how ImageGear implements its use:

DICOM images sometimes contain a Data Element called “Pixel Padding Value” (PPV). The PPV is used mostly to fill in the corners of round images. DICOM provides a Tag for PPV which is (0028,0120). This Data Element stores a 16-bit gray-scale value that is to be treated as the Pixel Padding Value. Any pixels in the image that have this value are not to be treated as meaningful objects—but as background color.

When ImageGear Medical loads a DICOM image that contains a PPV the value is captured and stored in the HDS, which is attached to the **HIGEAR** of the new image. In fact, 3 values are stored to the HDS: the PPV from the PPV Data Element, a flag indicating that a PPV was found in the file when it was loaded, and an 8-bit grayscale value to use to display pixels with this value. This function sets the values of these in-memory copies of the PPV data.

When **Use_Pix_Padding** is set to **TRUE**, pixels from the original image equal to the PPV are treated as background. All functions that fill the 16x8 LUT skip this value and place the **Show_PPV_As** value in the PPV slot of the table. This allows an application to adjust the contrast of the image while keeping the PPV or background constant. The background will be displayed with a gray-scale value

equal to that stored in **Show_PPV_As**. The PPV is also used for the IP functions. Functions like **MED_IP_min_max()** ignore pixel values that are equal to the PPV. (see below).

Use_Pix_Padding is initially set to **TRUE** if the loaded image contained the Pixel Padding Value Data Element (0028,0120). If this Data Element was not found then this defaults to **FALSE**

Pix_Padding_Val is initially set to the Data Field of the Pixel Padding Value Data Element (0028,0120) if it is found. If it is not it is set to default (**NULL**).

Show_PPV_As is not part of the PPV Data Element. This value always defaults to 64.

When the Pixel Padding Value is turned “on” (**Use_Pix_Padding = TRUE**) the **MED_IP_min_max()** function will know to ignore this value as it searches through the image for the brightest and darkest pixel value so that it avoids treating the Pixel Padding Value as the minimum or max pixel value. This error could easily occur because Pixel Padding Value is most often set to a very large or very small value so that it can be easily differentiated from the real pixel values.

To retrieve the Pixel Padding Value that is currently stored in the **HIGEAR** (not the one in the Data Set although them may have the same value) see **MED_DCM_DS_PixPadVal_get()**.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
LONG   Pix_Padding_Val;
BYTE   Show_PPV_as;
BOOL   Pref_use_pix_pad;

nErrcount = MED_DCM_DS_PixPadVal_set(hIGear, bPref_use_pix_pad,
Pix_Padding_Val, Show_PPV_as);
```

MED_DCM_DS_preamble_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_preamble_get(const HIGEAR
hIGear,
   LPCHAR *lpPreamble);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpPreamble	A far pointer to a buffer that will be used to hold the data from the preamble. The buffer must be at least 128 bytes.

Description:

This function gets the preamble item from the Part 10 header of the image, if one exists. Your receiving buffer must be at least 128 bytes.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_preamble_set

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_preamble_set(const HIGEAR
hIGear,
   LPCHAR      lpPreamble,
   const DWORD bytes_in_lpPreamble);
```

Arguments:

hIGear	**HIGEAR** handle to the image
lpPreamble	A far pointer to the preamble of the Data Set
bytes_in_lpPreamble	The number of bytes in your Preamble data

Description:

This function sets the value of the Preamble item of the Part 10 header. Provide it with the address of the Preamble in **lpPreamble**, and the number of bytes for the new Preamble in **bytes_in_lpPreamble**. If the length of the data is less than 128 bytes, the remainder of the Preamble is filled with **NULL**s.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_Rescale_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_Rescale_get(const HIGEAR hIGear,
   LPDOUBLE lpRescale_Slope,
   LPDOUBLE lpRescale_Intercept,
   LPBOOL   lpFound);
```

Arguments:

hIGear	Set to the **HIGEAR** handle of the image you would like to retrieve the Rescale values from.
lpRescale_slope	Returns you the value of the Rescale Slope Data Element as a **DOUBLE**. The Tag value of this DE is (0028,1053).
lpRescale_intercept	Returns you the value of the Rescale Intercept Data Element as a **DOUBLE**. The Tag value of this DE is (0028,1054).
lpFound	Returns whether or not these two Data Elements were found/ present in the image's Data Set.

Description:

This function will search the DICOM Data Set of the **HIGEAR** image and return the value from Rescale Slope (0028,1053) and Rescale Intercept (0028,1054). This function is a short cut that was created because the values of these Data Elements are often sought after. You could perform this same operation, as you would for getting the information from any DE, by using **MED_DCM_DS_move_find()** and **MED_DCM_DS_curr_data_get()**.

If both of these DEs are found in the Data Set, then **lpFound** returns **TRUE**. If one or both are missing then **lpFound** will return **FALSE**, **lpRescale_slope** will return 1.0 and **lpRescale_intercept** will return 0.0. **lpFound** can be set to **NULL** if you do not need to know if they are found or not. The returned slope and interface values returned are always usable even if they are not found in the Data Set.

Both this function and **MED_DCM_DS_Window_Level_get()** should be called before displaying an image using **MED_display_contrast()**.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR    hIGear;
DOUBLE RescaleSlope, RescaleIntercept;
```

```
nErrcount = MED_DCM_DS_Rescale_get(hIGear, &RescaleSlope,
&RescaleIntercept, NULL);
```

MED_DCM_DS_TAG_ELEMENT

Group: Data Set Utility

Group: (Macro)

Declaration:

```
MED_DCM_DS_TAG_ELEMENT(Tag);
```

Arguments:

Tag	Supply this argument with a 32-bit Tag value that begins with **DCM_TAG_**

Description:

This macro is used for getting the Element number out of a Tag. Supply it with a 32-bit Tag value and it will return you with the 16-bit Element Number value. The Element Number is the least significant **WORD** of the 32-bit Tag value. This macro is defined in **DataDict.h**.

Bits Per Pixel: N/A

Return Value:

Returns the 16-bit Element Number portion of a 32-bit Tag value.

Example:

```
element = MED_DCM_DS_TAG_ELEMENT(DCM_TAG_IssuerOfPatientID)
/* element equals 0x0021*/
```

MED_DCM_DS_TAG_GROUP

Group: Data Set Utility

Group: (Macro)

Declaration:

```
MED_DCM_DS_TAG_GROUP(Tag)
```

Arguments:

Tag	Supply this argument with a 32-bit Tag value

Description:

This macro is used for getting the Group Number out of a Tag. Supply it with a 32-bit Tag value and it will return you with the 16-bit Group Number value. The Group Number is the most significant **WORD** of the 32-bit Tag value. This macro is defined in **DataDict.h**.

Bits Per Pixel: N/A

Return Value:

Returns the 16-bit Group Number portion of a 32-bit Tag value

Example:

```
group = MED_DCM_DS_TAG_GROUP(DCM_TAG_IssuerOfPatientID)
/* group equals 0x0010*/
```

MED_DCM_DS_TAG_MAKE

Group: Data Set Utility

Group: (Macro)

Declaration:

```
MED_DCM_DS_TAG_MAKE(gn, en);
```

Arguments:

gn	Supply this with a valid DICOM Group Number
en	Supply this with a valid DICOM Element Number

Description:

This macro takes a 16-bit Group Number and a 16-bit Element Number and returns a 32-bit Tag value. The returned Tag is compatible with the **DCM_TAG_** constants in **DataDict.h**.

This macro is defined in **DataDict.h**.

Bits Per Pixel: N/A

Return Value:

Returns the 32-bit Tag value from the 16-bit Group Number and 16-bit Element Number that you provide.

Example:

```
AT_DCM_TAGTag;
Tag = MED_DCM_DS_TAG_MAKE(0x0010, 0x0021)
/* Tag equals DCM_TAG_IssuerOfPatientID or 0x00100021 */
```

MED_DCM_DS_TS_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_TS_get( const HIGEAR hIGear,
   LPAT_MODE lpTransfer_syntax);
```

Arguments:

hIGear	**HIGEAR** handle of the image
lpTransfer_syntax	Returns the Transfer Syntax constant that corresponds to the **DCM_PART10_ITEM_TRANSSYNTAXUID** part 10 item in the DataSet.

Description:

This function returns the Transfer Syntax constant that corresponds to the value of **DCM_PART10_ITEM_TRANSSYNTAXUID** part 10 item in the DataSet.

lpTransfer_syntax returns one of the Transfer Syntax constants, defined in **DataDict.h**, such as:

- **MED_DCM_TS_IMPLICIT_VR_LE**,
- **MED_DCM_TS_EXPLICIT_VR_LE**, etc.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_TS_set

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_TS_set(HIGEAR hIGear,
   const AT_MODE Transfer_syntax);
```

Arguments:

hIGear	**HIGEAR** handle of the image
Transfer_syntax	Set to the type of transfer syntax that you would like to store in the Transfer Syntax field of the Part 10 header.

Description:

This function sets the Transfer Syntax value in the Part 10 Header for “Transfer Syntax UID” Data Element (0002,0010). This value also overrides that set by **MED_DCM_DS_create()** function.

Transfer_syntax must be set to one of the following supported Transfer Syntax constants:

- **MED_DCM_TS_IMPLICIT_VR_LE,**
- **MED_DCM_TS_EXPLICIT_VR_LE**
- **MED_DCM_TS_EXPLICIT_VR_BE**

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_DS_update_file

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_update_file (HIGEAR hIGear,
const LPSTR lpszFileNameSrc, const LPSTR lpszFileNameDst);
```

Arguments:

hIGear	HIGEAR handle, containing metadata to be saved to result file
lpszFileNameSrc	Source file name
lpszFileNameDst	Result file name

Description:

This function creates a new file with exact copy of source file's pixel data and with new metadata (File Meta Information header and DataSet) taken from HIGEAR image handle. Pixel data is not decoded but copied directly from source to destination file.

This function can be used for updating metadata in an image file without modifying the pixel data. To achieve this, delete the source file after calling this function, and rename result file to the source file name.

The function does not change the tags that affect decoding of pixel data.

The function takes into account the following DICOM filter control parameters:

SAVE_GROUPLENGTHS

SAVE_ASPART10

MED_DCM_DS_Window_Level_get

Group: Data Set Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_Window_Level_get(const HIGEAR
hIGear,
   LPLONG lpWindow_Width,
   LPLONG lpWindow_Center,
   LPBOOL lpFound);
```

Arguments:

hIGear	**HIGEAR** handle to the image from which you would like to retrieve the Window Level values
lpWindow_Width	Returns the value of the Window Level Width Data Element (0028,1051).
lpWindow_Center	Returns the value of the Window Level Center Data Element (0028,1050)
lpFound	Returns a **TRUE** if both of these DEs are found in the Data Set; returns **FALSE** if one or the other is missing from the Data Set. Set to **NULL** if you do not need this information.

Description:

This function searches the Data Set of the **HIGEAR** image for the Window Width and Window Center Data Elements. This function is a short cut that was created because the values of these Data Elements are often sought after. You could perform this same operation, as you would for getting the information from any DE, by using **MED_DCM_DS_move_find()** and **MED_DCM_DS_curr_data_get()**.

If both are found they are returned and **lpFound** is set to **TRUE**. If one or both of these DEs are not found, **lpFound** is set to **FALSE** and ImageGear attempts to calculate adequate values for both Width and Center to display all pixels in the image.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
LONG   lWindow_min, lWindow_max;

nErrcount = MED_DCM_DS_Window_Level_get(hIGear, &lWindow_min,
&lWindow_max, NULL);
```

MED_DCM_load_DICOM

Group: Load/Save

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_load_DICOM(const LPSTR
lpszFileName,
   LPHIGEAR lphIGear,
   const AT_MODE nSyntax,
```

```
    const UINT page_number);
```

Arguments:

lpFileName	A far pointer to the name of the file to load
lphIGear	A far pointer which returns the **HIGEAR** handle for the newly loaded image
nSyntax	Set to the type of Transfer Syntax used for the file that you will be loading. Use one of the **MED_DCM_TS_** constants defined in **DCM.h**.
page_number	If the file is a multipage (multiframe) file, you may set this variable to specify the page to load. If the file is not multipage, set this to 1.

Description:

This function loads a DICOM image file. It cannot be used to load an image of any other format. The advantage to using this function over **IG_load_file()**is that it allows you more control, and is a faster loading utility because ImageGear's filter detection is set off for you.

This function loads the file specified by the path and filename in **lpFileName**, and returns you a new **HIGEAR** handle for the image in **lphIGear**.

To further speed up this function, provide it with the correct Transfer Syntax (TS) of the file. If you do not know the TS, ImageGear detects it for you if you set **nSyntax** to **MED_DCM_TS_AUTODETECT**.

You may also specify the page to load from a multipage (multiframe) file.

To remove a **HIGEAR** image from memory call **IG_image_delete()**.

NOTES: If your application supports unicode or multi-byte strings, you can open a file yourself ad pass us the FD handle. If you prefer to open your own file, use **MED_DCM_load_DICOM_FD()**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_load_DICOM_FD

Group: Load/Save

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_load_DICOM_FD(const AT_INT fd,
   const LONG    lOffset,
```

```
    LPHIGEAR      lphIGear,
    const AT_MODE nSyntax,
    const UINT    page_number);
```

Arguments:

fd	Set to the File Descriptor handle of the opened image file
lOffset	Set to a **LONG** value that gives the offset from the current position in the open file to the beginning of the DICOM data. In most cases this value will be 0.
lphIGear	A far pointer which returns the **HIGEAR** handle for the newly loaded image
nSyntax	Set to the type of Transfer Syntax used for the file that you will be loading. Use one of the **MED_DCM_TS_** constants defined in **DCM.h**.
page_number	If the file is a multipage file, you may set this variable to specify the page to load. If the file is not multipage, set this to 1.

Description:

This function performs the same operation as **MED_DCM_load_DICOM()** except that is takes a File Descriptor (FD) and offset instead of a filename. **lOffset** should be set to the position of the image within the file so that the ImageGear Medical knows where to begin decoding the image. If the beginning of the DICOM image file starts at the first byte, pass in an 0L for **lOffset**.

Since you are responsible for opening and closing the file this function can be used to bypass the **IG_load_file()** limitation of not handling unicode or multi-byte character strings.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_save_DICOM

Group: Load/Save

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_save_DICOM(const LPSTR
lpszFileName,
   const HIGEAR hIGear,
   const AT_MODE nSyntax,
   const BOOL bIncludeGroupLengths,
   const BOOL bSaveAsPart10,
   const AT_MODE PlanarConfiguration,
```

```
const BOOL IncludeSmallestLargest,
const UINT nJPEGQuality,
const DWORD dwReserved);
```

Arguments:

lpszFileName	A far pointer to the name of the file to save
hIGear	The **HIGEAR** handle of the image to save. This **HIGEAR** must have an attached DICOM Data Set.
nSyntax	Set to the type of Transfer Syntax to save the image with.
bIncludeGroupLengths	Set this Boolean variable to **TRUE** if you want to store the Group Lengths with the image
bSaveAsPart10	Set this Boolean variable to **TRUE** if you want to save a Part 10 header with the image
PlanarConfiguration	Determines how RGB pixel values are to be saved to the DICOM image file. The Planar Configuration Data Element (0028, 0006) will be automatically inserted and this value stored in it. If the Data Set already contains (0028,006) its value is ignored and the value of this parameter is used instead. This parameter is used for RGB only. It is ignored for all other image types. Set to one of the constants that begins with **MED_DCM_PLANAR_**.
IncludeSmallestLargest	If the image's original Data Set did not contain Data Elements for Smallest Image Pixel Value (0028,0106) and Largest Image Pixel Value (0028,0107) and you set **IncludeSmallestLargest** = **TRUE**, ImageGear will scan the image and determine a values for these DEs. Smallest Image Pixel Value and Largest Image Pixel Value will be included in the Data Set of the DICOM image being saved, and will contain the ImageGear-determined values. The values of the DE from the original Data Set (if any) will be ignored. If you set **IncludeSmallestLargest** = **FALSE**, ImageGear will not determine this value for you, and the Data Set of the image being saved will not include the Smallest Image Pixel Value and Largest Image Pixel Value Data Elements. However, if the original Data Set did contain these DEs, ImageGear will preserve and include them in the Data Set being saved.
nJPEGQuality	The value of this argument is only relevant to lossy JPEG compression. When using any other compression scheme, this value will be ignored. (This setting would be meaningless for Lossless JPEG compression). Set to the amount of pixel data to preserve during lossy JPEG compression. The range of valid values is 1-100 with a default value of 70. Higher settings result in higher quality and a larger file. Note that even at 100, JPEG compression is not capable of being completely "lossless." The compression used, if any, is set by **nSyntax**. (**MED_DCM_TS_JPEG_LOSSY_8** indicates the use of 8-bit JPEG compression).
dwReserved	Reserved for future use. Set to 0 for now.

Description:

This function saves a DICOM image file. It cannot be used to save an image of any other format. The advantage to using this function over **IG_save_file()**is that it allows you more control.

In order to save an image as a DICOM file the **HIGEAR** must have a Data Set attached to it. If you loaded the **HIGEAR** from a DICOM file then there already is a Data Set attached. If not, you can use the function **MED_DCM_DS_create()** to create one. If you are not sure if the image has a Data Set, call **MED_DCM_DS_exists()** to find out. If you are going to create a Data Set using **MED_DCM_DS_create()**, then you must still populate the Data Set with valid Data Elements using **MED_DCM_DS_DE_insert().**

The **nSyntax** parameter determines how the DICOM image file is encoded. The valid options are the constants whose names begin with **MED_DCM_TS_** (except for **_TS_UNKNOWN**, and **_TS_AUTODETECT**).

Group Lengths are optional in a DICOM image file. By default, Group Lengths are not stored in the Data Set that is attached to the **HIGEAR**. If you have read a DICOM file that included Group Lengths, they have been discarded as they were found. Set **bIncludeGroupLength** to **TRUE** to have them recreated and placed in the Data Set as it has been written to disk.

DICOM Image files are supposed to be written to disk using the Meta-Info Header that is defined in Part 10 of the DICOM Specification. However, many DICOM applications choose not to use this header. By setting **bSaveAsPart10** using **IG_image_control_set()** the Meta Information Header are placed at the beginning of the file. Setting this to **FALSE** will skip the header.

If the image is saved without a Part 10 header, it is often called a Raw DICOM image file. Note that the Part 10 Header Data Elements are not stored the same way as other Data Elements in the Data Set. If you wish to store this data you must populate the header using the **MED_DCM_DS_part10_set()** function before writing the file.

NOTE: If your application supports unicode or multi-byte strings, you can open a file yourself and pass us the FD handle. If you prefer to open your own file, use **MED_DCM_save_DICOM_FD()**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_save_DICOM_FD

Group: Load/Save

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_save_DICOM_FD(AT_INT fd,
    const HIGEAR hIGear,
    const AT_MODE nSyntax,
    const BOOL bIncludeGroupLengths,
    const BOOL bSaveAsPart10,
    const AT_MODE nPlanarConfiguration,
    const BOOL bInludeSmallestLargest,
```

```
        const UINT nJPEGQuality,
        const DWORD lReserved);
```

Arguments:

fd	Set to the FD handle of the open image. The image must have been opened with Read/Write access
hIGear	The **HIGEAR** handle of the image to save. This **HIGEAR** must have an attached DICOM Data Set.
nSyntax	Set to the type of Transfer Syntax used for the file that you will be saving. Use one of the **MED_DCM_TS_** constants defined in **DCM.h**.
bIncludeGroupLengths	Set this Boolean variable to **TRUE** if you want to store the Group Lengths with the image
bSaveAsPart10	Set this Boolean variable to **TRUE** if you want to save a Part 10 header with the image
nPlanarConfiguration	Determines how RGB pixel values are to be saved to the DICOM image file. The Planar Configuration Data Element (0028, 0006) will be automatically inserted and this value stored in it. If the Data Set already contains (0028,006) its value is ignored and the value of this parameter is used instead. This parameter is used for RGB only. It is ignored for all other image types. Set to one of the constants that begins with **MED_DCM_PLANAR_**.
bIncludeSmallestLargest	Setting this parameter to **TRUE** will cause the Smallest and Largest pixel value in the image to be computed and saved in Smallest Image Pixel Value (0028,0106) and Largest Image Pixel Value (0028,0107). If the Data Set contains these 2 Data Elements already they are ignored and the computed values are used. If this parameter is set to **FALSE** these two Data Elements are not stored in the file.
nJPEGQuality	The value of this argument is only relevant to lossy JPEG compression. When using any other compression scheme, this value will be ignored. Set to the amount of pixel data to preserve during lossy JPEG compression. The range of valid values is 1-100 with a default value of 70. Higher settings result is higher quality and a larger file. Note that even at 100, JPEG compression is not capable of being completely “lossless”.
lReserved	Reserved for future use. Set to 0 for now.

Description:

This function performs the same operation as **MED_DCM_save_DICOM()** except that it uses the FD instead of the file name. The file must have been opened with Read/Write access.

Since you are responsible for opening and closing the file this function can be used to bypass the **IG_save_file()** limitation of not handling unicode or multi-byte character strings.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_util_data_to_string

Group: DICOM Utility

Declaration:

```
BOOL ACCUAPI MED_DCM_util_data_to_string(const LPCHAR lpData,
   const  AT_DCM_VR vr,
   const  AT_DCM_VL vl,
   const  INT first_item,
   const  INT last_item,
   LPCHAR lpString,
   const  DWORD string_len,
   const  CHAR separator);
```

Arguments:

lpData	A far pointer to the data from a Data Element which you would like to convert to a string
vr	Set to the Value Representation (VR) for the Data. VR types have defined constants in **DCM.h** and begin with **MED_DCM_VR_**
vl	Set to the Value Length of the Data.
first_value	Set to the first Value to process. Set to -1 to process all values
last_vtem	Set to the last item to process. This argument is only effective if **first_value** is not set to -1.
lpString	A far pointer to a **NULL**-terminated string which returns the representation of the DICOM data field specified in **lpData**
string_len	Set this to the length in bytes of **lpString**. If the length is shorter than the data being retrieved, it will simply truncate the data.
separator	Set this to the character that you would like to use to separate multiple data values; for example, you might set it to a comma or semicolon.

Description:

This function takes the DICOM Data Field of the Current Data Element, designated in **lpData,** and converts it to a **NULL**-terminated character string.

Set the **vr** argument to determine how the **lpData** is to be interpreted.

lpString can only be filled up to **string_len - 1.** 1 is subtracted from the **string_len** to accommodate the **NULL**. Any remainder is simply clipped.

For Data Fields with more than one data value, the separator character specified in **separator** is used to delimit the items.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_util_tag_info_add

Group: DICOM Utility

Declaration:

```
BOOL ACCUAPI MED_DCM_util_tag_info_add(const AT_DCM_TAG Tag,
   const AT_DCM_VR VR,
   const AT_DCM_VM VM,
   const WORD      wVersion,
   const LPCHAR    lpszTagName);
```

Arguments:

Tag	Set to a Tag value. The Tag must be supplied as a 32-bit value in which the first 16 bits (WORD) represent the Group Number and the second 16 bits represent the Element Number. (See AT_DCM_TAG definition in DCM.h). Group and Element are expressed as WORDs.
lpVR	Set to the VR of the new Tag. The VR must be one of the predefined VRs that start with **MED_DCM_VR_** (see **DCM.h**).
lpVM	Set to the VM of the new Tag. Value Multiplicity tells whether and/or how many items can be stored in this type of Data Element. See Note below.
wVersion	Set this to the DICOM version. This should identify the first version of DICOM that includes this Tag. Most applications should set this to 3 but any value is accepted.
LpszTagName	Set to a character string name that should be provided as the description of this Tag.

Description:

This function allows to add new entries into an internal table of Tag entries. It can be used for adding newly defined DICOM Tags (new to the specification) or for adding private user-defined Tags. Once a new entry is added the new Tag works just like all other Tags do.

NOTES: Set **VM** to a non-zero positive integer for a Tag which must contain a specific number of Items. Set **VM** to **0** for a Tag which can have an unlimited number of Items. Set **VM** to a negative integer for a Tag which can have a limited number of items up to the absolute value of the provided **VM**. For example, **VM = -3** means that the Tag may have up to 3 items.

Bits Per Pixel: N/A

Return Value:

Returns **TRUE** if the new Tag was successfully added to the Data Dictionary; **FALSE** otherwise.

MED_DCM_util_tag_info_free

Group: DICOM Utility

Declaration:

```
BOOL ACCUAPI MED_DCM_util_tag_info_free(VOID);
```

Arguments:

None.

Description:

This function frees up the user-defined Data Dictionary, if it exists. All User-Defined entries are discarded. The pre-defined Data Dictionary is not affected. This function returns **TRUE** if a table was freed. **FALSE** if there was no table to free.

Bits Per Pixel: N/A

Return Value:

TRUE if a table was freed; **FALSE** if there was no table to free.

MED_DCM_util_tag_info_get

Group: DICOM Utility

Declaration:

```
BOOL ACCUAPI MED_DCM_util_tag_info_get(const AT_DCM_TAG Tag,
   LPAT_DCM_VR lpVR,
   LPAT_DCM_VM lpVM,
   LPWORD      lpwVersion,
   LPCHAR      lpszTagName);
```

Arguments:

Tag	Set to a Tag value. The Tag must be supplied as a 32-bit value in which the first 16 bits (WORD) represent the Group Number and the second 16 bits represent the Element Number. (See **DCM.h**).
lpVR	A far pointer which returns the current VR (Value Representation). Set to **NULL** if you don't need this information. **AT_DCM_VR** is a structure defined in **DCM.h**.
lpVM	A far pointer which returns the VM (Value Multiplicity) of the current Tag; set to **NULL** if you don't need this information. Value Multiplicity tells you whether and/or how many items can be stored in this type of Data Element. See Note below.
lpwVersion	Returns the version of DICOM Specification, such as 3.0
lpszTagName	Returns the name of the specified Tag.

Description:

This function returns information about the specified Tag. It returns the Value Representation, Value Multiplicity, DICOM version, and the Tag name. The function searches in the user-defined Data Dictionary first, and then if the Tag is not found, searches in the static Data Dictionary, which represents the Data Dictionary listed in Part 6 of the DICOM standard.

You might use this function before making a call that alters a Data Element.

NOTES: If **lpVM** returns a Positive integer: there must be this number of Items. If it returns a 0: you may have an unlimited number of items, including 0. If it returns a Negative integer: The number of items may include up to the absolute value of the value returned. For example, if **lpVM = -3**, you may have up to 3 items.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_util_VR_info_mode

Group: DICOM Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_util_VR_info_mode(const AT_DCM_VR
vr_mode,
    LPCHAR lpVRstring,
    LPWORD lpwLength,
    LPWORD lpwRestriction,
```

```
LPBOOL lpCheck_form,
LPBOOL lpIsString);
```

Arguments:

vr_mode	Set this to the type of VR (or data type) you would like information on. Use one of the **MED_DCM_VR_** constants defined in **DCM.h**
lpVRstring	A far pointer that returns the text representation of VR in 3 characters
lpwLength	A far pointer that returns the size of the VR
lpwRestriction	A far pointer that returns any restriction flags These will be returned as constants that are defined in **DCM.h** and begin with **MED_DCM_LEN_**. See below.
lpReserved	This argument has not been implemented yet. Please set to **NULL** for now.
lpIsString	A far pointer to a **BOOL** value that tells you whether the data of this type of VR is a **NULL**-terminated string or not. If **TRUE,** data of this type is a **NULL**-terminated string and could be printed using the printf format %s. If **FALSE,** the data is an integer or other binary data type.

Description:

This function looks up the Value Representation specified in **vr_mode** and returns the following information about it: its text representation, length, restrictions, and whether or not it can be a **NULL**-terminated string.

These are the possible restriction flags that can be returned to you:

- **MED_DCM_LEN_MAX -** Up to this maximum
- **MED_DCM_LEN_FIXED -** Fixed Length
- **MED_DCM_LEN_TS -** Depends on Transfer Syntax
- **MED_DCM_LEN_NA -** Not applicable
- **MED_DCM_LEN_UNLIMITED -** No max length

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_DCM_util_VR_info_string

Group: DICOM Utility

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_util_VR_info_string(const LPCHAR
lpVR_string,
   LPWORD lpwLength,
   LPWORD lpwRestriction,
   LPBOOL lpCheck_form,
   LPBOOL lpIsString,
   LPAT_DCM_VR lpVr);
```

Arguments:

lpVRstring	Set to the 3-character text representation of the VR as used in the DICOM Specification, e.g. “PN”, “CS”, or “SQ.” While these are 2-character codes, the end of line string termination makes it 3.
lpwLength	A far pointer that returns the size of the VR. This is the length in byes of a single instance of data that is of the specified type. For example, a VR of “SL” has a fixed length of 4 bytes per item. If the VM allows more than a single data value, then each one will take up 4 bytes. Other VRs have a maximum length. “PN” and “UI” both have a maximum of 64 bytes.
lpwRestriction	A far pointer that returns any restriction flags. These will be returned as constants that are defined in **DCM.h** and begin with **MED_DCM_LEN_**. See below.
lpReserved	This argument has not been implemented yet. Please set to **NULL** for now.
lpIsString	A far pointer to a **BOOL** value that tells you whether the data of this type of VR is a **NULL**-terminated string or not. If **TRUE,** data of this type is a **NULL**-terminated string and could be printed using the printf format %s. If **FALSE,** the data is an integer or other binary data type.
lpVR	A far pointer that returns an **AT_MODE** constant that identifies the type of Value Representation. Such constants begin with **MED_DCM_VR_**.

Description:

This function returns the type of Value Representation used. You supply it with a 3-character string (two alphabetic characters plus the end-of-string terminator), and it will return you one of the **MED_DCM_VR_** constants defined in **DCM.h**.

These are the possible restriction flags that can be returned to you:

- **MED_DCM_LEN_MAX -** Up to this maximum
- **MED_DCM_LEN_FIXED -** Fixed Length

- **MED_DCM_LEN_TS -** Depends on Transfer Syntax
- **MED_DCM_LEN_NA -** Not applicable
- **MED_DCM_LEN_UNLIMITED -** No max length

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_display_color_create

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_color_create(const AT_MODE
scheme,
   const LONG param1,
   const LONG param2,
   const LONG param3,
   LPAT_PIXEL lpRLUT,
   LPAT_PIXEL lpGLUT,
   LPAT_PIXEL lpBLUT);
```

Arguments:

scheme	Set to the pre-defined color scheme that you would like to use to pseudocolor an image. These **AT_MODE** color schemes are defined in **MedAPI.h** and begin with **DCM_PSEUDOCOLOR_SCHEME_**. See below for details.
param1	Set to a **LONG** value if the color scheme requires it. See below.
param2	Set to a **LONG** value if the color scheme requires it. See below.
param3	Set to a **LONG** value if the color scheme requires it. See below.
lpRLUT	Pass this argument an array of 256 bytes. It returns the Red LUT that will be used to pseudocolor an image when **MED_display_color_set()** is called.
lpGLUT	Pass this argument an array of 256 bytes. It returns the Green LUT that will be used to pseudocolor an image when **MED_display_color_set()** is called.
lpBLUT	Pass this argument an array of 256 bytes. It returns the Blue LUT that will be used to pseudocolor an image when **MED_display_color_set()** is called.

Description:

This function is used to fill 3 LUTs with one of several pseudocolor schemes. The scheme parameter selects which ImageGear pre-defined pseudocolor method to use. Some of the schemes may require parameters that are passed into param1, param2, and param3. Other schemes do not use these parameters and any value passed in are ignored. See below.

lpRLUT, **lpGLUT**, and **lpBLUT** must be passed an array of 256 bytes, each. When this function returns, these 3 tables will be filled with values that can be used to pseudocolor an 8—16 bit gray-scale image.

This function does not apply the color to an image. It only fills these 3 LUTs. To apply these tables to an image you need to follow this function call with a call to **MED_display_color_set()**.

As of this writing, these are the available preset ImageGear pseudocolor schemes:

DCM_PSEUDOCOLOR_SCHEME_OFF	Reset to grayscale - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_1	Oil Film - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_2	Dark Blue to Bright Red - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_3	Green to Red - does not use param1, param2, param3

DCM_PSEUDOCOLOR_SCHEME_4	Red, Green, Blue - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_5	Thermal - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_6	Bright Rainbow - does not use Param 1,2,3

Bits Per Pixel: 8—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;
{
AT_PIXEL RLUT[256];
AT_PIXEL GLUT[256];
AT_PIXEL BLUT[256];

MED_display_color_create(DCM_PSEUDOCOLOR_SCHEME_6, 0, 0, 0,
RLUT, GLUT, BLUT);

MED_display_color_set(hIGear, RLUT, GLUT, BLUT);

repaint_image_and_error_check(hWnd);
}
```

MED_display_color_limits

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_color_limits(HIGEAR hIGear,
   const INT thresh_low,
   const INT low_red,
   const INT low_green,
   INT       low_blue,
   const INT thresh_high,
   const INT high_red,
   const INT high_green,
   const INT high_blue);
```

Arguments:

hIGear	**HIGEAR** handle to the image
thresh_low	Set to an **INT** value for the lower pixel value limit. All pixels at or below this value will be pseudocolored.
low_red	8—16 Set to an INT for the red component of the RGB color that will be used for all pixels below **thresh_low**
low_green	Set to an INT for the green component of the RGB color that will be used for all pixels below **thresh_low**
low_blue	Set to an INT for the blue component of the RGB color that will be used for all pixels below **thresh_low**
thresh_high	Set to an INT value for the upper pixel value limit. All pixels at or above this value will be pseudocolored.
high_red	Set to an INT for the red component of the RGB color that will be used for all pixels above **thresh_high**
high_green	Set to an INT for the green component of the RGB color that will be used for all pixels above **thresh_high**
high_blue	Set to an INT for the blue component of the RGB color that will be used for all pixels above **thresh_high**

Description:

This function can be used to pseudocolor the brightest and the darkest pixels in a 16-bit gray-scale image. This is typically used to clearly see pixel values that are over-saturated (255 and up) or under-saturated (0 and below). However, this function lets you customize the settings for the upper and lower pixel value limits with **thresh_low** and **thresh_high**.

The color is applied after the 16x8 display LUT is applied to the image and is not affected by altering this LUT.

Pixels having values at or below **thresh_low** are colored by **low_color** and pixels having values at or above **thresh_high** are colored by **high_color**. All pixels having values within the 2 limits are set to normal linear gray.

To turn off this effect, set **thresh_low** to -1 and **thresh_high** to 256.

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use **IG_display_image()**.

Bits Per Pixel: 9—16

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_display_color_set

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_color_set(HIGEAR hIGear,
   const LPAT_PIXEL lpRLUT,
   const LPAT_PIXEL lpGLUT,
   const LPAT_PIXEL lpBLUT);
```

Arguments:

hIGear	**HIGEAR** handle of the 8—16-bit gray-scale image to color
lpRLUT	Set to a 256 byte Red table or set to **NULL** for ramp
lpGLUT	Set to a 256 byte Green table or set to **NULL** for ramp
lpBLUT	Set to a 256 byte Blue table or set to **NULL** for ramp

Description:

This function allows applications to apply pseudocolor to 8—16-bit gray-scale images. The 3 LUTs can be user-defined or they can be filled by calling **MED_display_color_create()**. Each LUT must either be 256 bytes each or set to **NULL**. If a **NULL** is passed in as one of the LUTs, a linear 0—255 gray-scale LUT is used. If a **NULL** is not passed in, the LUT points to an array of 256 bytes. The entries from this LUT are used to color the output of the 16x8 LUT.

The pseudocoloring is not altered by any functions that update the 16x8 LUT. This function alters the display of a gray-scale image only and does not change the pixel values of the image or any entries in the 16x8 LUT.

To turn off the pseudocolor, simply pass in **NULLs** for all of the LUT parameters. These three color channels will be reset to their default linear ramps.

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use **IG_display_image()**.

Bits Per Pixel: 8—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
{
AT_PIXEL RLUT[256];
AT_PIXEL GLUT[256];
AT_PIXEL BLUT[256];
```

```
MED_display_color_create(DCM_PSEUDOCOLOR_SCHEME_6, 0, 0, 0,
RLUT, GLUT, BLUT);
MED_display_color_set(hIGear, RLUT, GLUT, BLUT);

repaint_image_and_error_check(hWnd);

}
```

MED_display_contrast

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_contrast(HIGEAR hIGear,
    const DOUBLE rescale_slope,
    const DOUBLE rescale_intercept,
    const LONG window_center,
    const LONG window_width,
    const DOUBLE gamma);
```

Arguments:

hIGear	16g image having its 16x8 LUT updated
rescale_slope	Set to the desired value for Rescale Slope (0028,1053). Call **MED_DCM_DS_Rescale_get()** to obtain this value.
rescale_intercept	Set to the desired value for Rescale Intercept (0028,1054). Call **MED_DCM_DS_Rescale_get()** to obtain this value.
window_center	Set to the desired value for Window Center (0028,1050). Call **MED_DCM_DS_Window_Level_get()** to obtain this value.
window_width	Set to the desired value for Window Width (0028,1051). Call **MED_DCM_DS_Window_Level_get()** to obtain this value.
gamma	Set this to the amount of Gamma correction you would like applied to the image. To turn off Gamma correction, set to 1.0. The valid range of values is any **DOUBLE** between 0.20—1.80.

Description:

This function uses the window/level mapping method to change the contrast of an 9—16-bit image. It takes values for Window Center & Width, and Rescale Slope & Intercept, and it also allows you to apply Gamma correction, if you desire.

This function fills the 16x8 LUT with values that will display a 9-16 bit image according to VOI LUT (Window Center/Width) and Modality LUT (Rescale Intercept/Slope) values that you specify. Any values that have been in the 16x8 LUT are overwritten with these new ones.

If the Rescale Slope and Intercept of the image are known, you should provide them. If they are not known, pass in a 1.0 and 0.0, respectively.

If you do not have values for Window Center and Window Width, these can be calculated using the minimum and maximum pixel values (if you have them known). Here are the formulas you should use to display all pixel values:

Window Center = (max + min) / 2;

Window Width = (max - min);

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use **`IG_display_image()`**.

See also **`MED_display_contrast_auto()`**

Bits Per Pixel: 9—16

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_display_contrast_auto

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_contrast_auto(const HIGEAR
hIGear,
   const LPAT_RECT lpRect,
   const DOUBLE     rescale_slope,
   const DOUBLE     rescale_intercept,
   const LONG       lReserved_option,
   LPLONG           lpWindow_center,
   LPLONG           lpWindow_width);
```

Arguments:

hIGear	The **HIGEAR** handle to the image to convert
lpRect	Use this **AT_RECT** structure to specify the rectangular portion of the image for which to optimize the contrast on; set to **NULL** for the whole image. Please see the *ImageGear User's Manual* if you are unfamiliar with this structure.
rescale_slope	Set to the value of the Data Element, Rescale Slope (0028,1053). You can use **MED_DCM_DS_Rescale_get()** to obtain this value. If this Data Element is not present in the Data Set, please set this value to 1.0.
rescale_intercept	Set to the value of the Data Element, Rescale Intercept (0028,1054). You can use **MED_DCM_DS_Rescale_get()** to obtain this value. If this Data Element is not present in the Data Set, please set this value to 0.0.
lReserved_option	Reserved for future use; please set to 0 for now
lpWindow_center	A far pointer that returns a **LONG** for the Window Center; set to **NULL** if you don't need this information.
lpWindow_width	A far pointer that returns a **LONG** for the Window Width; set to **NULL** if you don't need this information.
lpSigned	A far pointer that returns a **BOOL** value indicating whether the value is signed or not.

Description:

This function automatically fills the 16x8 LUT of a 16-bit image loaded into ImageGear in order to optimize its displaying. The rectangular portion of the image that you specify (or the whole image) is scanned and the maximum and minimum pixel values are determined; the Window Center and Width are calculated from these.

If the Rescale Slope and Intercept of the image are known, you should provide them. If they are not known, pass in a 1.0 and 0.0, respectively.

The Window Center and Width that this function computes are passed back to you in **lpWindow_center** and **lpWindow_width**, unless you set these arguments to **NULL**.

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use **IG_display_image()**.

See also **MED_display_contrast().**

NOTES: Since this function must scan the pixels for the minimum and maximum values, it takes more time to run than **IG_display_contrast()**.

Bits Per Pixel: 9—16

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example:

```
HIGEAR hIGear;
DOUBLE fRescaleSlope = 1.0;
DOUBLE fRescaleIntercept = 1.0;
DOUBLE fGama = 1.0;
LONG   WindowWidth;
LONG   WindowCenter;

MED_display_contrast_auto(hIGear, NULL, fRescaleSlope,
          fRescaleIntercept, fGamma, 0, &WindowCenter,
                                                &WindowWidth);
```

MED_display_grayscale_LUT_build

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_grayscale_LUT_build(
   AT_MED_DCM_DISPLAY_SETTINGS *lpDICOMDisplaySettings,
   HIGLUT hLUT
   );
```

Arguments:

lpDICOMDisplaySettings	AT_MED_DCM_DISPLAY_SETTINGS structure.
hLUT	LUT handle.

Description:

This function builds a grayscale LUT according to `lpDICOMDisplaySettings`.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_display_grayscale_LUT_build_auto

Group: Display

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_grayscale_LUT_build_auto(
   HIGEAR hIGear,
   const AT_RECT *lpRect,
   AT_MED_DCM_DISPLAY_SETTINGS *lpDICOMDisplaySettings,
   HIGLUT hlut
   );
```

Arguments:

hIGear	16g image having its 16x8 LUT updated
lpRect	AT_RECT structure.
lpDICOMDisplaySettings	AT_MED_DCM_DISPLAY_SETTINGS structure.
hlut	LUT handle.

Description:

This function calculates `lpDICOMDisplaySettings->VOIWindow` from image's min and max values, and then builds grayscale LUT according to `lpDICOMDisplaySettings`.

Bits Per Pixel: 8...16-bit

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_contrast

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_contrast(HIGEAR hIGear,
   const DOUBLE rescale_slope,
   const DOUBLE rescale_intercept,
   const LONG   window_center,
   const LONG   window_width,
   const DOUBLE gamma);
```

NOTES: The functionality of **MED_IP_window_level()** has been incorporated into the new function **MED_IP_contrast()**, which also takes settings for Rescale Slope, Rescale Intercept, and gamma correction

Arguments:

hIGear	16g image that will have its 16x8 LUT updated
rescale_slope	Set to the desired value for Rescale Slope (0028,1053). Call **MED_DCM_DS_Rescale_get()** to obtain this value.
rescale_intercept	Set to the desired value for Rescale Intercept (0028,1054). Call **MED_DCM_DS_Rescale_get()** to obtain this value.
window_center	Set to the desired value for Window Center (0028,1050). Call **MED_DCM_DS_Window_Level_get()** to obtain this value.
window_width	Set to the desired value for Window Width (0028,1051). Call **MED_DCM_DS_Window_Level_get()** to obtain this value.
gamma	Set this to the amount of Gamma correction you would like applied to the image. To turn off Gamma correction, set to 1.0. The valid range of values is any **DOUBLE** between 0.20—1.80.

Description:

This function converts a 16-bit gray-scale image to an 8-bit gray-scale image in the same way as **MED_display_contrast()**. The difference is that this function permanently alters the pixel values.

See also **MED_display_contrast()**

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_contrast_auto

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_contrast_auto(const HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const DOUBLE    rescale_slope,
   const DOUBLE    rescale_intercept,
   const LONG      lReserved_option,
   LPLONG          lpWindow_center,
   LPLONG          lpWindow_width);
```

Arguments:

hIGear	The **HIGEAR** handle to the image to convert
lpRect	Use this **AT_RECT** structure to specify the rectangular portion of the image for which to optimize the contrast on; set to **NULL** for the whole image. Please see the *ImageGear User's Manual* if you are unfamiliar with this structure.
rescale_slope	Set to the value of the Data Element, Rescale Slope (0028,1053). You can use **MED_DCM_DS_rescale_get()** to obtain this value. If this Data Element is not present in the Data Set, please set this value to 1.0.
rescale_intercept	Set to the value of the Data Element, Rescale Intercept (0028,1054). You can use **MED_DCM_DS_rescale_get()** to obtain this value. If this Data Element is not present in the Data Set, please set this value to 0.0.
lReserved_option	Reserved for future use; please set to 0 for now
lpWindow_center	A far pointer that returns a **LONG** for the Window Center; set to **NULL** if you don't need this information.
lpWindow_width	A far pointer that returns a **LONG** for the Window Width; set to **NULL** if you don't need this information.
lpSigned	A far pointer that returns a **BOOL** value indicating whether the value is signed or not.

Description:

This function converts a 16-bit gray-scale image to an 8-bit gray-scale image using the same function as **MED_display_contrast_auto()** except that this function permanently alters the pixel values.

See also **MED_display_contrast_auto()**, and **MED_IP_reduce_depth_with_downshift()**.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_high_bit_transform

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_high_bit_transform(HIGEAR hIGear,
   const LONG lMin);
```

Arguments:

hIGear	**HIGEAR** handle of the image
lMin	New High bit value

Description:

This function changes the High Bit Data Element of the currently loaded DICOM 16-bit gray-scale image. The 16-bits are shifted up or down as needed to accommodate the new High Bit value, and the High Bit Data Element is updated. Zeros are shifted in as needed.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_histo_clear

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_histo_clear(LPLONG lpHisto,
   const DWORD nBin_count,);
```

Arguments:

lpHisto	A far pointer of type **LONG** which points to the buffer that will was used to hold the histogram
nBin_count	Set this **DWORD** variable to the number of bins allocated for the histogram

Description:

This function is used to clear the histogram created by **MED_IP_histo_tabulate()**.

Bits Per Pixel: N/A

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_histo_tabulate

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_histo_tabulate(const HIGEAR hIGear,
   const LPAT_RECT lpRect,
   LPLONG    lpHisto,
   const WORD nBin_width,
   const DWORD dwBin_count,
   LPBOOL    lpSigned,
   LPLONG    lpCount);
```

Arguments:

hIGear	The **HIGEAR** handle to the image to create a histogram for
lpRect	A far pointer to a struct of type **AT_RECT** that defines the rectangular portion of the image to use for creating a histogram. Please see the *ImageGear User's Manual* if you are unfamiliar with this structure.
lpHisto	A far pointer to a buffer to be used for holding the histogram
nBin_width	An integer variable that specifies the range of pixel values to be counted into one bin
dwBin_count	A **DWORD** variable that specifies the number of bins allocated
lpSigned	A far pointer to a **BOOL** that returns the sign status of the image. If it returns **TRUE**, the image is signed.
lpCount	A far pointer to a **LONG** which returns the number of pixels counted.

Description:

This function is used to tabulate the histogram of a 8 or 16-bit gray-scale image. The histogram is returned in the memory block, **lpHisto**, allocated by the application. This memory block must be large enough to hold the histogram for the image. Each histogram bin must be 4 bytes wide. The size of **lpHisto** needed (in bytes) can be computed as follows:

size = ((possible_pixel_values) * 4) / nBin_width

where **possible_pixel_values** depends on the bit depth of the image (8g=256, 9g=512,...). **nBin_width** is used to determine the range of pixel values that are counted in their own bin. A value of 1 indicates that each pixel value is counted in its own histogram bin. A value of 2 would allow neighboring values (such as 128 and 129) to be counted as a single bin (as a single value). If **nBin_width==possible_pixel_values** then only a single histogram bin (4 bytes) is filled and the count will be equal to the number of pixel values in the **lpRect**.

nBin_count is used as a safety. This should be set to the number of bins in the histogram that your application has allocated. If a pixel value is going to overflow this memory it will be ignored. This pixel value will not be included in the sum returned in **lpCount** (you can use this to determine if any values were ignored).

Bits Per Pixel: 8—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_min_max

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_min_max(const HIGEAR hIGear,
   const LPAT_RECTlpRect,
   LPLONG    lpMin,
   LPLONG    lpMax,
   LPBOOL    lpSigned);
```

Arguments:

hIGear	The **HIGEAR** handle of the image to scan
lpRect	Use this **AT_RECT** structure to specify the rectangular portion of the image to scan. Please see the *ImageGear User's Manual* if you are unfamiliar with this structure.
lpMin	A far pointer to a **LONG** which returns the value of the minimum pixel value in the region that was "scanned"
lpMax	A far pointer to a **LONG** which returns the value of the maximum pixel value in the region that was "scanned"
lpSigned	A far pointer to a **BOOL** which returns the status of the image sign. **TRUE** means the image is signed.

Description:

This function scans an 8- or 16-bit gray-scale image and returns the raw minimum and maximum pixel values and the "is signed" flag. If the image is 16-bit signed then the returned values are also signed.

Note that this function returns the min and max **raw** pixel values. That is, the returned values are not corrected using the Modality LUT (Rescale Slope/ Intercept) values. To apply this correction to the min and max values that you get from **MED_IP_min_max()** call the function **MED_DCM_DS_Rescale_get()** which will return you the values of Rescale Slope and Rescale Intercept. Then use the following formulas:

min_corrected = (min_raw * rescale_slope) + rescale_intercept;

max_corrected = (max_raw * rescale_slope) + rescale_intercept;

When a 16-bit gray-scale image is “unsigned” it has pixel values between 0 and 65,000. If a 16-bit image is “signed” it has pixel values between -32k and +32k. By the same rule, when an 8-bit gray-scale image is “unsigned” it has pixel values between 0-255. If an 8-bit image is “signed”, it has pixel values between -128 and +127. Some modalities of DICOM use signed images. So if you know whether the image is signed or unsigned will help you to interpret the minimum and maximum values.

Bits Per Pixel: 8—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Example

```
HIGEAR hIGear;
AT_ERRCOUNT nErrcount;
AT_RECT rcROI;
LONG lMin, lMax;
BOOL bSigned;

nErrcount = MED_IP_min_max(hIGear, &rcROI, &lMin, &lMax,
&bSigned);
```

MED_IP_normalize

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_normalize(HIGEAR hIGear,
   const LONG lMin);
```

Arguments:

hIGear	The **HIGEAR** handle to the image to normalize
lMin	A variable of type **LONG** used to set the minimum pixel value for the resulting image

Description:

This function's main purpose is to convert the pixel data of a 16-bit image from signed to unsigned. If you are using it for this reason, set **lMin** to 0.

It can also be used to convert the minimum pixel value for the image. To do this, set **lMin** to greater than 0. This function searches the image for the minimum pixel value. It maps this value to **lMin**. Then, all pixel values are linearly adjusted to maintain the original contrast. If **lMin** is equal to the min pixel value in **hIGear** then no change is made. The resulting image remains a 16-bit.

Bits Per Pixel: Input image: 8—16g; Output image: 8—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_promote_to_16_gray

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_promote_to_16_gray(HIGEAR hIGear,
   const UINT iBits,
   const UINT iHighBit);
```

Arguments:

hIGear	The **HIGEAR** handle of the image to be converted
iBits	An integer variable that sets the number of bits that are actually used. The value range is 9 - 16 bits.
iHighBit	An integer variable that sets the high bit for the new 16-bit image. The eighth bit of the original 8 bits is positioned here. If the image is saved as a DICOM, this information is saved to the High Bit Data Element (0028,0102).

Description:

This function takes an 8-bit gray-scale or color image and converts it to a gray-scale image with a bit depth of 16 bpp. The original 8-bit image is discarded. All images created by this function have 16-bit pixels, **iBits** sets the number of bits out of the 16 that are actually used. **iHighBit** sets the new position among the 16 bits at which the original 8 bits should be positioned.

Bits Per Pixel: Input: 8g; Output: 9—16

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_reduce_depth_with_downshift

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_reduce_depth_with_downshift (HIGEAR
hIGear,
   const UINT downshift);
```

Arguments:

hIGear	**HIGEAR** handle to the image
downshift	A **UINT** variable specifying the

Description:

This function is used to downshift a chosen range of 8 bits (out of a maximum of 16) to the 256 pixel values that can be shown on an 8-bit display device. For example, if you set **iBits** to 8, and the image has 16 bpp, bits 8-15 will be downshifted and used alone as the pixel values.

This function will directly alter the pixel data.

See also **MED_IP_reduce_depth_with_LUT()**.

Bits Per Pixel: 9—16 input; 8g output

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_reduce_depth_with_LUT

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_reduce_depth_with_LUT(HIGEAR hIGear,
   const LPBYTE lpLUT,
   const DWORD dwEntries);
```

Arguments:

hIGear	The **HIGEAR** handle to the image to convert
lpLUT	A far pointer to the lookup table to use for reduction. Set to 0 to use the display LUT. Set to **NULL** if you want to use the display LUT.
dwEntries	A variable of type **DWORD** that specifies the number of entries in the LUT. This value is ignored if the display LUT is used.

Description:

This function takes a 16-bit gray-scale image and reduces it to a 8-bit gray-scale one using provided (filled) LUT or current 16x8 display LUT. The 16-bit image is discarded and replaced with the new 8-bit gray-scale image.

To reduce memory requirements the LUT does not have to have a 16-bit input. **dwEntries** should hold the number of entries in the LUT. If there is a pixel found that can overflow the LUT it is ignored and replaced with a 0.

NOTES: This function does not reduce 8-bit images. To reduce an 8-bit image, use the appropriate **`IG_IP_color_reduce_…()`** function from the baseline ImageGear API.

NOTES: Once this function is called, the display LUT will be thrown away, since it is now an 8-bit image.

See also **`MED_IP_reduce_depth_with_downshift()`**.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_IP_swap_bytes

Group: Image Processing

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_swap_bytes(HIGEAR hIGear);
```

Arguments:

hIGear	**`HIGEAR`** handle of the image

Description:

This function swaps the 2 bytes of each 16-bit pixel in the image. It can be used to help fix poorly constructed images which have the high and low bytes of each 16-bit gray-scale pixel reversed. All pixels in the image are transformed.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_modality_transform_apply

Group: Modality

Declaration:

```
AT_INT MED_modality_transform_apply(
    const AT_MED_MODALITY_RESCALE *lpRescale,
    HIGLUT hModalityLUT,
    AT_INT Value
    );
```

Arguments:

lpRescale	AT_MED_MODALITY_RESCALE structure
hModalityLUT	Modality LUT handle.
Value	Value to apply Modality transform to.

Description:

If hModalityLUT is not NULL, applies Modality LUT to the specified value. Otherwise, applies the linear modality transform (lpRescale) to the specified value.

Bits Per Pixel: 9—16g

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_overlay_apply

Group: Overlay

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_overlay_apply(HIGEAR hIGear);
```

Arguments:

hIGear	Page from which to load overlay planes into ART marks

Description:

Loads overlay planes from DICOM metadata (DataSet) attached to the image and creates ART marks of type IMAGE_EMBEDDED for each overlay plane. Each mark is placed in a separate ART group.

Overlay planes do not support specifying the colors (even black or white). An overlay pixel can only be visible or invisible. All visible overlay pixels are displayed with the same color, which is specified by the application and not stored in the DICOM DataSet. By default, ImageGear uses yellow color to display overlays. The application can change overlay color by going through all marks in the HIGEAR and changing the image palette of IMAGE_EMBEDDED marks.

This function requires the ART component to be attached to ImageGear.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

See Also:

ARTX Component API Function Reference

MED_overlay_extract

Group: Overlay

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_overlay_extract(HIGEAR hIGear);
```

Parameters:

hIGear	Page for which to save annotations into DataSet as overlay planes.

Description:

Saves ART marks into DICOM dataset as overlay planes. Each ART group is saved into a separate overlay plane.

Since overlay planes do not support any types of graphic objects other than bitonal bitmaps, all ART marks from a group are converted into a bitmap before saving into an overlay.

Overlay planes do not support specifying the colors (even black or white). An overlay pixel can only be visible or invisible. All visible overlay pixels are displayed with the same color, which is specified by the application and not stored in the DICOM DataSet.

This function requires the ART component to be attached to ImageGear.

See Also:

ARTX Component API Function Reference

MED_PS_apply

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_apply (
HIGEAR hIGear,
HIGEAR hIGearPresState,
DWORD dwFeatureFlags,
DWORD dwGrpID,
LPAT_MED_LUT_DESC lpGSDFLUT
)
```

Arguments:

hIGear	Image to which Pres State shall be applied
hIGearPresState	Image containing Presentation State DataSet
dwFeatureFlags	Tells which features of PS to use – bit mask
dwGrpID	Display Group ID
lpGSDFLUT	GSDF LUT info

Description:

This function extracts PS data from DICOM DataSet of hIGearPresState, and adjusts display of hIGear based on this data. This function may result in the following:

- Adjust Display LUT associated with the image, using
  - VOI and Modality LUTs found in the presentation state DataSet
  - Presentation LUT found in the presentation state DataSet
  - GSDF LUT parameter
- Change display layout, including image rectangle, zoom and orientation
- Add ART objects

If there are no VOI or Modality LUT in the hIGearPresState, the function tries to find them in the DataSet of hIGear. If there are no such LUTs in hIGear, default values are taken.

If MED_PS_FEATURE_PRES_LUT option is specified, the function assumes that MED_PS_FEATURE_CONTRAST is selected as well.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_display_contrast

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_display_contrast (
HIGEAR hIGear, // 16g image to have its LUT16x8 updated
DOUBLE rescale_slope, // Rescale Slope (0028,1053)
DOUBLE rescale_intercept, //
LONG window_center, //
LONG window_width, //
DOUBLE gamma, //
LPAT_MED_LUT_DESC lpPresLUTInfo, //
LPAT_MED_LUT_DESC lpGSDFLUTInfo, //
)
```

Arguments:

hIGear	16g image to have its LUT16x8 updated
rescale_slope	Rescale Slope (0028,1053)
rescale_intercept	Rescale Intercept (0028,1054)
window_center	Window Center (0028,1050)
window_width	Window Width (0028,1051)
gamma	Gamma correction - set to 1.0 to turn off
lpPresLUTInfo	Presentation LUT info
lpGSDFLUTInfo	GSDF LUT info

Description:

Builds the 16x8 display LUT, using rescale, window, Presentaion LUT and GSDF LUT.

If `lpGSDFLUTInfo` parameter is not `NULL`, gamma is not used.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_display_contrast_auto

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_display_contrast_auto (
HIGEAR hIGear,
LPAT_RECT lpRect,
DOUBLE rescale_slope,
DOUBLE rescale_intercept,
DOUBLE gamma,
LPAT_MED_LUT_DESC lpPresLUTInfo,
LPAT_MED_LUT_DESC lpGSDFLUTInfo,
LPLONG lpWindow_center,
LPLONG lpWindow_width
)
```

Arguments:

hIGear	16g image to have its LUT16x8 updated
lpRect	set to NULL to scan the entire image
rescale_slope	Rescale Slope (0028,1053)
rescale_intercept	Rescale Intercept (0028,1054)
gamma	Gamma correction - set to 1.0 to turn off
lpPresLUTInfo	Presentation LUT info
lpGSDFLUTInfo	GSDF LUT info
lpWindow_center	Window Center (0028,1050)
lpWindow_width	Window Width (0028,1051)

Description:

Builds the 16x8 display LUT, using rescale, Presentaion LUT and GSDF LUT. Window center and width are calculated automatically, based on min and max values of the image. If `lpGSDFLUTInfo` parameter is not `NULL`, gamma is not used.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_extract

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_extract (
HIGEAR hIGear,
HIGEAR hIGearPresState,
DWORD dwFeatureFlags,
DWORD dwGrpID
)
```

Arguments:

hIGear	Image which settings should be exported to PS
hIGearPresState	Image that will contain Presentation State DataSet
dwFeatureFlags	Tells which features of PS to export – bit mask
dwGrpID	Display Group ID

Description:

This function adds Presentation State tags to the DataSet of hIGearPresState, based on the display settings of `hIGear`, an ART marks attached to it. If any of the tags are already present in the `hIGearPresState`, they are overwritten.

This function may add/update the following tags/sequences in the `hIGearPresState`:

- DCM_TAG_DisplayedAreaSelectionSequence
- DCM_TAG_GraphicAnnotationSequence
- DCM_TAG_GraphicLayerSequence
- DCM_TAG_ImageHorizontalFlip
- DCM_TAG_ImageRotation

To create a new presentation state DataSet, use the following steps:

1. Call IG_image_create_DIB_ex() with lCompression set to IG_BI_EMPTY
2. Call MED_PS_extract() to add Presentation State tags
3. Add the other tags required by the Presentation State module using general DataSet access functions.
4. Save Presentation State to file using IG_fltr_save_file() or similar functions

To modify an existing Presentation State DataSet, use the following steps:

1. Load a Presentation State DataSet using `IG_fltr_load_file()` or similar functions
2. Apply the Presentation State to a DICOM image, using `MED_PS_apply()`
3. Change display settings or ART marks
4. Extract Presentation Data from DICOM image back to Presentation State DataSet.
5. Save Presentation State to file using `IG_fltr_save_file()` or similar functions

Use `MED_PS_pres_LUT_set()` to add Presentation LUT to the Presentation State `HIGEAR`.

Use general DataSet functions to add VOI and Modality LUT to the Presentation State `HIGEAR`.

Use IG image saving functions (`nFormat=IG_FORMAT_DCM`) to save `lphIGearPresState` into a presentation state file (.pre).

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_GSDF_LUT_build

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_GSDF_LUT_build(
DWORD dwCharactCurveEntryCount,
LPDOUBLE lpCharactCurve,
LPAT_MED_LUT_DESC lpLUT
)
```

Arguments:

dwCharactCurveEntryCount	Number of entries in Charactistic Curve
lpCharactCurve	Characteristic Curve of the display
lpLUT	GSDF LUT to fill

Description:

This function builds a look up table that maps pixel intencities from DICOM GSDF-compliant color space into pixel intencities of the display device. The user should fill in the members of the `lpLUT` structure and allocate the `lpLUTData` buffer. The space necessary for holding the LUT can be calculated using the `AM_MED_LUT_SIZE_GET` macro.

The LUT obtained from this function can be used for building the 16x8 or 8x8 display LUT. See `MED_PS_import()`, `MED_display_contrast_PS()`.

At the moment, ImageGear does not support any display devices that are capable of displaying more than 256 shades of grayscale. Hence, `MED_PS_GSDF_LUT_build` can only build a LUT consisting of 8-bit entries.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_GSDF_LUT_init

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_GSDF_LUT_init(
DWORD     dwCharactCurveEntryCount,
LPDOUBLE  lpCharactCurve,
HIGLUT    lut
);
```

Arguments:

dwCharactCurveEntryCount	Number of entries in Charactistic Curve
lpCharactCurve	Characteristic Curve of the display
lut	GSDF LUT to fill.

Description:

This function initializes the `HIGLUT` object with GSDF LUT data.

This function replaces **`MED_PS_GSDF_LUT_build()`**.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_pres_LUT_get

Group: Presentation State

Declaration:

```
MED_PS_pres_LUT_get(
HIGEAR hIGearPresState,
LPAT_MED_LUT_DESC lpPresLUTInfo
);
```

Arguments:

`hIGearPresState`	Image that contains Presentation State DataSet
`lpPresLUTInfo`	Presentation LUT info

Description:

This function fills the Presentation LUT with the data contained in the Presentation State DataSet.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_pres_LUT_info_get

Group: Presentation State

Declaration:

```
MED_PS_pres_LUT_info_get(
HIGEAR hIGearPresState,
LPAT_MED_LUT_DESC lpPresLUTInfo
);
```

Arguments:

hIGearPresState	Image that contains Presentation State DataSet
lpPresLUTInfo	Presentation LUT info

Description:

This function returns information about Presentation LUT contained in a Presentation State DataSet.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_pres_LUT_set

Group: Presentation State

Declaration:

```
MED_PS_pres_LUT_set(
HIGEAR hIGearPresState,
LPAT_MED_LUT_DESC lpPresLUTInfo
);
```

Arguments:

hIGearPresState	Image that contains Presentation State DataSet
lpPresLUTInfo	Presentation LUT info

Description:

This function adds Presentation LUT sequence to the DataSet of hIGearPresState. Use this function together with `MED_PS_extract()`.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_PS_pres_state_GSDF_apply

Group: Presentation State

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_pres_state_GSDF_apply(
HIGEAR hIGear,
HIGEAR hIGearPresState,
DWORD dwFeatureFlags,
DWORD dwGrpID,
HIGLUT GSDFLUT
);
```

Arguments:

hIGear	Image to which Pres State shall be applied.
hIGearPresState	Image containing Presentation State DataSet.
dwFeatureFlags	Tells which features of PS to use - bit mask.
dwGrpID	Display Group ID.
GSDFLUT	GSDF LUT.

Description:

This function extracts PS data from DICOM DataSet of `hIGearPresState`, and adjusts display of `hIGear` based on this data. This function may result in the following:

- Adjust Display LUT associated with the image, using
  - VOI and Modality LUTs found in the presentation state DataSet
  - Presentation LUT found in the presentation state DataSet
  - GSDF LUT parameter
- Change display layout, including image rectangle, zoom and orientation
- Add ART objects

If there are no VOI or Modality LUT in `hIGearPresState`, the function tries to find them in the DataSet of `hIGear`. If there are no such LUTs in `hIGear`, default values are taken.

If `MED_PS_FEATURE_PRES_LUT` option is specified, the function assumes that `MED_PS_FEATURE_CONTRAST` is selected.

This function replaces **MED_PS_apply()**. It provides the same functionality as **MED_PS_apply()**, but uses a more general type HIGLUT for GSDFLUT parameter.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

MED_VOI_window_init_from_min_max

Group: Window

Declaration:

```
AT_VOID MED_VOI_window_init_from_min_max(
    AT_MED_VOI_WINDOW *lpWindow,
    AT_INT Min,
    AT_INT Max
    );
```

Arguments:

lpWindow	Window structure.
Min	Window min.
Max	Window max.

Description:

This function initializes a **AT_MED_VOI_WINDOW** structure from window Min and Max.

Return Value:

None.

MED_VOI_window_max_get

Group: Window

Declaration:

```
AT_INT MED_VOI_window_max_get(
    AT_MED_VOI_WINDOW *lpWindow
    );
```

Arguments:

lpWindow	Window structure.

Description:

This function returns **AT_MED_VOI_WINDOW** maximum.

Return Value:

AT_MED_VOI_WINDOW maximum.

MED_VOI_window_min_get

Group: Window

Declaration:

```
AT_INT MED_VOI_window_min_get(
    AT_MED_VOI_WINDOW *lpWindow
    );
```

Arguments:

lpWindow	Window structure.

Description:

This function returns **AT_MED_VOI_WINDOW** minimum.

Return Value:

AT_MED_VOI_WINDOW minimum.

AT_MED_DCM_DISPLAY_SETTINGS

Group: Structure

Declaration:

```
struct AT_MED_DCM_DISPLAY_SETTINGS
{
    AT_MED_MODALITY_RESCALE ModalityRescale;
    HIGLUT ModalityLUT;
    AT_MED_VOI_WINDOW VOIWindow;
    HIGLUT VOILUT;
```

```
    AT_BOOL IsInverted;
    AT_DOUBLE Gamma;
    HIGLUT PresentationLUT;
    HIGLUT GSDFLUT;
    AT_MED_PIXEL_PADDING_SETTINGS PixelPadding;
};
typedef struct AT_MED_DCM_DISPLAY_SETTINGS;
```

Arguments:

ModalityRescale	Specifies linear Modality transform (Rescale Slope/Intercept). Ignored if ModalityLUT is not NULL.
ModalityLUT	Specifies Modality LUT.
VOIWindow	Specifies linear VOI transform (Window center/width). Ignored if VOI LUT is not NULL.
VOILUT	Specifies VOI LUT.
IsInverted	Tells whether the image should be displayed inverted. TRUE Corresponds to MONOCHROME1 photometric interpretation, FALSE coresponds to MONOCHROME2 photometric interpretation.
Gamma	Specifies gamma correction value. Ignored if GSDFLUT is not NULL.
PresentationLUT	Specifies Presentation LUT.
GSDFLUT	Specifies Grayscale Standard Display Function LUT.
PixelPadding	Specifies pixel padding settings.

Description:

This structure contains settings that can be used for displaying DICOM images with proper contrast. The settings are applied in the following order: Modality (LUT or Rescale), VOI (LUT or Window), IsInverted flag, Presentation LUT, Gamma or GSDFLUT.

AT_MED_MODALITY_RESCALE

Group: Structure

Declaration:

```
struct AT_MED_MODALITY_RESCALE
{
    AT_DOUBLE Slope;
    AT_DOUBLE Intercept;
};
typedef struct AT_MED_MODALITY_RESCALE;
```

Arguments:

Slope	Rescale slope
Intercept	Rescale Intercept

Description:

This structure specifies linear Modality transform.

AT_MED_PIXEL_PADDING_SETTINGS

Group: Structure

Declaration:

```
struct AT_MED_PIXEL_PADDING_SETTINGS
{
    AT_BOOL UsePixPadding;
    AT_INT PixPaddingValue;
    AT_INT ShowPaddingAs;
};
```

Arguments:

UsePixPadding	Specifies whether pixel padding should be taken into account during display.
PixPaddingValue	Specifies pixel padding value.
ShowPaddingAs	Specifies the output intensity for displaying padding pixels.

Description:

This structure represents DICOM pixel padding settings.

AT_MED_VOI_WINDOW

Group: Structure

Declaration:

```
struct AT_MED_VOI_WINDOW
{
    AT_INT Center;
    AT_INT Width;
};
```

```
typedef struct AT_MED_VOI_WINDOW;
```

Arguments:

Center	Window center
Width	Window width

Description:

This structure specifies linear VOI transform.

PDF Component API Function Reference

The ImageGear PDF component is responsible for the PDF functionality. The ImageGear PDF component exposes handles and objects described in the following table. The general PDF layer is implemented via “PDF” objects (for example HIG_PDF_DOC). The PDF editing layer is implemented via “PDE” objects (for example HIG_PDE_CONTENT).

NOTE: This chapter links to the PDF reference (Adobe PDF 1.5 specification), which can be downloaded from here: http://partners.adobe.com/asn/tech/pdf/specifications.jsp

NOTE: The Adobe®-based PDF Component is applicable for Linux (i386), Solaris 32-bit, and AIX Unix Platforms.

Linux for PowerPC, HP-UX 32-bit & 64-bit, HP-UX for Itanium, and Sun Solaris 64-bit use the GS PDF Component. See the GS PDF Component API section.

To initialize the ImageGear PDF component use the function **`IG_PDF_initialize()`**.

To terminate the ImageGear PDF component use the function **`IG_PDF_terminate()`**.

Summary

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

General API	
`C API`	DLL API which provides general ImageGear PDF Component functionality.
Basic Objects	

HIG_PDF_BASOBJ	Basic Object – basic PDF object interface. PDF supports eight basic types of object: ■ Boolean values ■ Integer and Fixed (real) numbers ■ Strings ■ Names ■ Arrays ■ Dictionaries ■ Streams ■ The null object Objects may be labeled so that they can be referred to by other objects. A labeled object is called an indirect object.
HIG_PDF_BASBOOL	Basic Boolean – PDF provides boolean objects identified by the keywords true and false. Boolean objects can be used as the values of array elements and dictionary entries.
HIG_PDF_BASINT	Basic Integer – Integer objects represent mathematical integers within a certain interval centered at 0.
HIG_PDF_BASFIXED	Basic Fixed – Fixed objects approximate mathematical real numbers, but with limited range and precision; they are typically represented in fixed-point, rather than floating-point, form.
HIG_PDF_BASSTR	Basic String – A string object consists of a series of bytes—unsigned integer values in the range 0 to 255. The string elements are not integer objects, but are stored in a more compact format.
HIG_PDF_BASNAME	Basic Name – A name object is an atomic symbol uniquely defined by a sequence of characters. Uniquely defined means that any two name objects made up of the same sequence of characters are identically the same object. Atomic means that a name has no internal structure; although it is defined by a sequence of characters, those characters are not “elements” of the name.
HIG_PDF_BASARR	Basic Array – an array object is a one-dimensional collection of objects arranged sequentially. Unlike arrays in many other computer languages, PDF arrays may be heterogeneous; that is, an array’s elements may be any combination of numbers, strings, dictionaries, or any other objects, including other arrays.
HIG_PDF_BASDICT	Basic Dictionary – a dictionary object is an associative table containing pairs of objects, known as the dictionary’s entries. The first element of each entry is the key and the second element is the value. The key must be a name. The value can be any kind of object, including another dictionary.
HIG_PDF_BASNULL	Basic Null – The null object has a type and value that are unequal to those of any other object. There is only one object of type null, denoted by the keyword **null**.

General Objects	
HIG_PDF_ACTION	Handle to a PDF action object, which is a task that performs when a user clicks on a link or a bookmark.
HIG_PDF_ATOM	Atom - a hashed token used in place of strings to optimize performance (it is much faster to compare Atoms than strings). Many methods use Atoms.
HIG_PDF_BOOKMARK	Handle to a PDF bookmark object, which allows the user to navigate interactively from one part of the document to another. It consists of a tree-structured hierarchy of bookmarks.
HIG_PDF_STREAM	Stream - a data stream that may be a buffer in memory, or an arbitrary user-written procedure. Typically used to extract or provide data.
HIG_PDF_DESTINATION	Handle to a PDF destination object, which represents a particular view of a page in a document. It contains a reference to a page, a rectangle on that page, and information specifying how to adjust the view to fit the window's size and shape.
HIG_PDF_DICTIONARY	Handle to a PDF dictionary object, which represents an optional-content membership dictionary object.
HIG_PDF_DOC	Document - the underlying PDF representation of a document. Through PDF Document, your application can perform most of the Edit Pages operations (delete, replace, and so on). Thumbnails can be created and deleted through this object. You can set and retrieve document information fields through this object as well.
HIG_PDF_FONT	Handle to a PDF font object that is used to draw text on a page. A HIG_PDF_FONT has a number of attributes, including an array of widths, the character encoding.
HIG_PDF_LAYER	Handle to a PDF layer object, which represents a named object whose state can be toggled in a UI to affect changes in visibility of content.
HIG_PDF_PAGE	Page - a single page in the PDF representation of a document. A page contains a series of objects representing the objects drawn on the page (Graphic), a list of resources used in drawing the page, annotations (Annotation), an optional thumbnail image of the page, and the beads used in any articles that occur on the page.
HIG_PDF_WORDFINDER	WordFinder – extracts words from a PDF file, and enumerates the words on a single page or on all pages in a document..
HIG_PDF_WORD	Word - a word in a PDF file. Each word contains a sequence of characters in one or more styles (see Style).
HIG_PDF_STYLE	Style - provides access to information about the fonts, font sizes, and colors used in a Word.

HIG_PDF_SYSFONT	SysFont – a reference to a font installed in the host system. SysFont methods allow you to list the fonts available in the host system and to find a font in the system that matches a PDE Font, if it is present.
HIG_PDF_SYSENCODING	SysEncoding – provides system encoding for a PDF file.
Page Editing Objects and Elements	
HIG_PDE_OBJECT	Object – base interface for all the PDE objects.
HIG_PDE_CONTENT	Content – modifiable content of a PDF page, which contains elements. Content may be obtained from an existing page or from a Form XObject.
HIG_PDE_COLORSPACE	ColorSpace – a reference to a color space used on a page in a PDF file. The color space is part of the graphics state attributes of a PDE Element.
HIG_PDE_FONT	Font – A reference to a font used on a page in a PDF file. It may be equated with a font in the system.
HIG_PDE_ELEMENT	Element - base interface for the elements of a page display list (PDE content) and for clip objects. The general PDE element methods allow you to get and set general element properties.
HIG_PDE_CLIP	Clip – a list of elements containing a list of Paths and Texts that describe a clip state. Clips can be created and built up with PDE Clip methods. Any PDE Element object can have Clip associated with it. Clip objects can contain PDE Containers and PDE Groups to an arbitrary level of nesting. This allows PDE Containers to be used to mark clip objects. PDE Groups inside PDE Clips that contain at least one PDE Text and no PDE Paths have a special meaning. All PDE Text objects contained in such a PDE Group are considered to be part of the same BT/ET block. This means that the union of these PDE Texts makes up a single clipping path—as opposed to the intersection of the PDE Texts. See Section 5.3 in the PDF reference for more information about BT/ET block.
HIG_PDE_CONTAINER	Container - a group of elements on a page in a PDF file. In the PDF file, containers are delimited by Marked Content BMC/EMC or BDC/EMC pairs. Every container has a Marked Content tag associated with it. In addition to grouping a set of elements, a BDC/EMC pair specifies a property list to be associated with the grouping. Thus a container corresponding to a BDC/EMC pair also has a property list dictionary associated with it. See Section 10.5 in the PDF reference for more information about Marked Content operators.

HIG_PDE_FORM	Form - an element that corresponds to an instance of XObject Form on a page (or other containing stream such as another XObject Form or annotation form). The context associated with this instance includes the actual stream that represents the XObject Form and the initial conditions of the graphics state. The latter consists of the transformation matrix, initial color values, and so forth. It is possible to have two Forms that refer to the same XObject Form. The forms will exist at different places on the same page, depending on the transformation matrix. They may also have different colors or line stroking parameters. In the case of a transparency group, the opacity is specified in the gstate. Within a Form, each element has its own gstate (or is a container, place, or group object). These gstates are independent of the parent Form gstate. Form elements may have their own opacity. Content may be obtained from a Form to edit the form's display list.
HIG_PDE_GROUP	Group - an in-memory representation of objects in Content. It has no state and is not represented in any way in a content stream (that is, Content). When used in a Clip, this object is used to associate Text objects into a single clipping object.
HIG_PDE_IMAGE	Image - an element that contains an Image XObject or in-line image. You can associate data or a stream with an image.
HIG_PDE_PATH	Path – an element that contains a path. Path objects can be stroked, filled, and/or serve as a clipping path.
HIG_PDE_PLACE	Place – an element that marks a place on a page in a PDF file. In a PDF file, a place is represented by the MP or DP Marked Content operators. Marked content is useful for adding structure information to a PDF file. For instance, a drawing program may want to mark a point with information, such as the start of a path of a certain type. Marked content provides a way to retain this information in the PDF file. A DP operator functions the same as the MP operator and, in addition, allows a property list dictionary to be associated with a place.
HIG_PDE_POSTSCRIPT	PostScript – an element representing in-line or XObject pass-through PostScript object. XObject PostScripts are listed in page XObject resources.
HIG_PDE_SOFTMASK	SoftMask– an object for creating and manipulating a soft mask in a PDF file.
HIG_PDE_SHADING	Shading – an element that represents smooth shading.

HIG_PDE_TEXT	Text – an element representing text. It is a container for text as show strings or as individual characters. Each sub-element may have different graphics state properties. However, the same clip applies to all sub-elements of a Text. Also, the char path of a Text can be used to represent a clip.
HIG_PDE_TEXTITEM	TextItem – a PDE element representing a text object.
HIG_PDE_XGROUP	XGroup – a transparency (XGroup) resource.
HIG_PDE_XOBJECT	XObject – an element representing an arbitrary XObject.
Subsidiary Types / DLL Structures	
AT_PDF_BOOL	Boolean type with two values: TRUE (1) or FALSE (0)
AT_PDF_FIXED	The Fixed type is a 32-bit quantity representing a rational number with the high (low on little-endian machines) 16 bits representing the number's mantissa and the low (high) 16 bits representing the fractional part. The definition is platform-dependent. LPAT_PDF_FIXED is a pointer to an AT_PDF_FIXED. Addition, subtraction, and negation with AT_PDF_FIXED types can be done with + and -, unless you care about overflow. (Overflow in Fixed-value operations is indicated by the values IG_PDF_FIXED_POSITIVE_INFINITY and IG_PDF_FIXED_NEGATIVE_INFINITY.)
AT_PDF_FIXEDPOINT	Point (in two-dimensional space) represented by two fixed numbers.
AT_PDF_FIXEDMATRIX	Matrix containing fixed numbers.
AT_PDF_FIXEDRECT	A rectangle represented by the coordinates of its four sides. A rectangle differs from a quadrilateral in that the former must always have horizontal and vertical sides, and opposite sides must be parallel.
AT_PDF_FIXEDQUAD	Quadrilateral represented by four fixed points (one at each corner). A quadrilateral differs from a rectangle in that the latter must always have horizontal and vertical sides, and opposite sides must be parallel.
AT_PDF_FONT_STYLES	Font Styles.
AT_PDF_FONT_METRICS	Font Metrics.
AT_PDF_PRINTOPTIONS	Used to control printing.
AT_PDE_FONTATTRS	Attributes of a PDE Font and a PDF SysFont.
AT_PDE_COLORVALUE	Color value.
AT_PDE_COLORSPEC	Color specification.
AT_PDE_DASH	Dash specification, as described in Table 4.8 in the PDF Reference. See Section 4.3.2 for more information on line dash patterns.

AT_PDE_GRAPHICSTATE	Attributes of a PDE Element or a PDE Text sub-element.
AT_PDE_TEXTSTATE	Attributes of a PDE Text element.
AT_PDF_SYSFONT_PLATDATA	PDF SysFont platform specific data.
AT_PDE_FONT_INFO	PDE Font information.
AT_PDE_FONT_CREATEFROMSYSFONTPARAMS	Parameters for PDE font creation.
AT_PDE_COLORRANGE	Color range.
AT_PDE_XYZCOLOR	XYZ color.
AT_PDE_COLORDATA_CALGRAY	CalGray color space.
AT_PDE_COLORDATA_CALRGB	CalRGB color space.
AT_PDE_COLORDATA_LAB	L*a*b* color space.
AT_PDE_COLORDATA_ICCBASED	ICC based color space.
AT_PDE_COLORDATA_INDEXED	Indexed color space.
AT_PDE_COLORDATA_SEPARATION	Separation color space
AT_PDE_COLORDATA_DEVICEN	DeviceN color space.
AT_PDE_COLORDATA	Fill in one of the following members of this union, then pass it to PDE ColorSpace creation routine. Please see section 7.10 of the PDF Reference Manual for information on color spaces.
AT_PDE_PSATTRS	Attributes of a PDE PS object.
AT_PDE_IMAGEATTRS	Attributes of a PDE Image object.
AT_PDE_FILTERSPEC	Filter element in a filter array.
AT_PDE_FILTERARRAY	Filter information for streams. Array of FilterSpec elements. Usually consists of 2 filter elements: text encoding and image compression.
AT_PDE_CONTENTATTRS	Attributes of a PDE Content object.

General API

General ImageGear PDF API calls, macros and callback functions.

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

General API	
C API	DLL API which provides general ImageGear PDF Component functionality.

C API

Type: Control

Description:

DLL API that provides general ImageGear PDF Component functionality. These general Functions, Macros and Callbacks are used for several different tasks: initialize and terminate the the PDF Component, fixed-point math utilities, Callbacks integrated with ImageGear Core components.

Class Members:

IG_PDE_get_default_gstate	Function	Fills out a LPAT_PDE_GRAPHICSTATE structure with the default graphic state.
IG_PDF_get_host_encoding	Function	Indicates what kind of host encoding a system uses.
IG_PDF_register_authproc	Function	Registers the authorization callback
IG_PDF_text_extract	Function	This function extracts text from pages determined by arguments nStartPage and nPageCount.
IG_PDF_translate_to_host	Function	Translates a string from Unicode or PDFDocEncoding to host encoding.
IG_PDF_translate_to_pdf	Function	Translates a string from host encoding to PDFDocEncoding or Unicode.
AM_PDF_LONG_TO_FIXED	Macro	Converts **x** to AT_PDF_FIXED and returns it.
AM_PDF_FIXED_ROUND_TO_LONG	Macro	Converts the AT_PDF_FIXED number **f** to an integer, rounding it to the nearest long integer value and returns it.
AM_PDF_FIXED_TRUNC_TO_LONG	Macro	Converts the AT_PDF_FIXED number **f** to an integer, truncating it to the next lower long integer value and returns it.
AM_PDF_SHORT_TO_FIXED	Macro	Converts **x** to AT_PDF_FIXED and returns it.
AM_PDF_FIXED_ROUND_TO_SHORT	Macro	Converts the AT_PDF_FIXED number **f** to an integer, rounding it to the nearest short integer value and returns it.
AM_PDF_FIXED_TRUNC_TO_SHORT	Macro	Converts the AT_PDF_FIXED number **f** to an integer, truncating it to the next lower short integer value and returns it.
AM_PDF_DOUBLE_TO_FIXED	Macro	Converts **x** to AT_PDF_FIXED and returns it.
AM_PDF_FIXED_TO_DOUBLE	Macro	Converts the AT_PDF_FIXED number **f** to double value and returns it.
LPFNIG_PDF_AUTHPROC	Callback	Callback used by document open routine. It is called when an encrypted document is being opened, to determine whether or not the user is authorized to open the file.
LPFNIG_PDF_STREAM_PROC	Callback	Callback for use by **IG_PDF_stream_read_CB_register** and **IG_PDF_stream_write_CB_register**. This procedure must return the number of bytes specified by nDataLen, obtaining them in any way it wishes.

LPFNIG_PDF_STREAM_DESTROYPROC	Callback	Callback for use by **IG_PDF_stream_write_CB_register**. Called at end of stream so you can do clean up and free allocated memory.
LPFNIG_PDF_SYSFONT_ENUMPROC	Callback	Callback for **IG_PDF_sysfont_enumerate**. It is called once for each system font.
LPFNIG_PDE_CLIP_ENUMPROC	Callback	Callback for **IG_PDE_clip_enumerate_elements**, which enumerates all of a PDE Clip's PDE Elements in a flattened manner.

IG_PDE_get_default_gstate

Group: C API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_get_default_gstate(
   LPAT_PDE_GRAPHICSTATE lpGstate
   )
```

Parameters:

lpGstate	Pointer to AT_PDE_GRAPHICSTATE structure with the default graphic state.

Description:

Fills out a LPAT_PDE_GRAPHICSTATE structure with the default graphic state.

NOTE: Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this method. Be sure to release these non-NULL objects when disposing of lpGstate.

Return Value:

Error count

IG_PDF_get_host_encoding

Group: C API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_get_host_encoding(
   LPVOID *lpHostEncoding
```

)

Arguments:

lpHostEncoding	Returns 0 for a Roman system; nonzero for a non-Roman system (a structure that depends on the host encoding). Users should simply test whether this value is 0 or not.

Description:

Indicates what kind of host encoding a system uses. Allows you to determine whether a system is Roman or non-Roman. (Non-Roman is also known as CJK-capable, that is, capable of handling multi-byte character sets, such as Chinese, Japanese, or Korean.)

Host encoding is a platform-dependent encoding for the host machine. For non-UNIX Roman systems, it is MacRomanEncoding in Mac OS and WinAnsiEncoding in Windows. In UNIX (except HP-UX) Roman systems, it is ISO8859-1 (ISO Latin-1); for HP-UX, it is HP-ROMAN8. See Appendix D in the PDF Reference for descriptions of MacRomanEncoding, WinAnsiEncoding, and PDFDocEncoding.

For non-Roman systems, the host encoding may be a variety of encodings, which are defined by a CMap (character map). See Section 5.6.4 in the PDF Reference for a list of predefined CMaps.

Use **`IG_PDF_get_host_encoding`** to determine if a system's host encoding is Roman or not.

Return Value:

Error count

IG_PDF_initialize

Group: C API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_initialize(
LPVOID lpVoid);
```

Arguments:

lpVoid	Reserved, must be set to NULL.

Description:

This function is used to initialize the ImageGear PDF Component. To initialize the ImageGear PDF component it needs to be attached to the core ImageGear and then **`IG_PDF_initialize()`** function must be called.

The following resource content is required by the ImageGear PDF component initialization routine:

Resource\PDF\CIDFont\	PDF CID fonts directory
Resource\PDF\CMap\	PDF font CMaps directory
Resource\PDF\Font\	PDF fonts directory
Resource\PS\ColorRendering\	color rendering PostScript
Resource\PS\ICCProfiles\	Directory containing the ICC profiles that allow using the Adobe® Color Engine® (ACE®) **NOTE:** The profiles in this directory must be placed in the system folder named, which on Windows is named **`\Windows\System32\Color`**.
Resource\PS\Fonts\	PS fonts directory
Resource\PS\ProcSet\	PostScript procedures
Resource\PS\ps.vm	a file for initializing the PostScript Interpreter's virtual memory
Resource\PS\startupNORM.ps	startup PostScript program used to initialize the PostScript Interpreter
Resource\PS\superatm.db	Adobe® Type Manager® (ATM®) database used to substitute missing fonts

The ImageGear PDF component uses the following PDF global control parameters to locate the resource content:

PDF.PDF_RESOURCE_PATH	path to the Resource\PDF directory
PDF.PS_RESOURCE_PATH	path to the Resource\PS directory
PDF.HOST_FONT_PATH	path to the system font directory
PDF.TMP_PATH	path to the TEMP directory

If the PDF global parameters are not defined, the ImageGear PDF Component behavior depends on the OS:

- Resource content gets from the ImageGear component directory defined by **`COMM.PATH`** parameter
- Host fonts get from the Windows font directory
- Temporary directory gets from the **`GetTempPath()`** result

Return Value:

Error count

IG_PDF_register_authproc

Group: API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_register_authproc(
    LPFNIG_PDF_AUTHPROC lpfnAuthProc,
    LPVOID lpAuthData
    );
```

Arguments:

lpfnAuthProc	Authorization callback, called only if the file has been secured (that is, if the file has either the user or the master password set). This callback should obtain whatever information is needed to determine whether the user is authorized to open the file, then call IG_PDF_doc_perm_request (which returns the permissions that the authentication data enables).
lpAuthData	Pointer to user-supplied data to pass to lpfnAuthProc each time it is called.

Description:

Registers the authorization callback, which will be called when open a secured PDF file, i.e. a PDF that has either the user or the master password set.

Return Value:

Error count

IG_PDF_terminate

Group: C API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_terminate();
```

Description:

This function terminates the ImageGear PDF component, and must be called before terminating a user application.

NOTE: For a multi-threaded application, you must call IG_PDF_initialize and IG_PDF_terminate in the thread - not in the calling module.

Return Value:

Error count

IG_PDF_text_extract

Group: API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_text_extract(
LPSTR lpszFileName,
LPSTR lpszTextName,
UINT nStartPage,
UINT nPageCount);
```

Arguments:

lpFileName	Name of the PDF or PS document
lpTextFileName	Name of the output TXT file
nStartPage	Number of the first page
nPageCount	Total number of pages to be processed (starting at `nStartPage`)

Description:

This function extracts text from pages determined by arguments nStartPage and nPageCount.

Return Value:

Error count

IG_PDF_translate_to_host

Group: C API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_translate_to_host(
    LPCSTR szInPDFStr,
    LONG nInPDFStrSize,
    LPSTR szOutHostStr,
    LONG nOutHostStrSize,
    LPLONG lpnOutHostStrBytes
    );
```

Arguments:

szInPDFStr	Pointer to the string to translate (may point to the same memory as szOutHostStr, allowing strings to translate in place).
nInPDFStrSize	The length of szInPDFStr, in bytes.
szOutHostStr	Pointer to the translated string (may point to the same memory as szInPDFStr).
nOutHostStrSize	The length of the szOutHostStr buffer, in bytes.
lpnOutHostStrBytes	Number of bytes in the translated string szOutHostStr.

Description:

Translates a string from Unicode or PDFDocEncoding to host encoding. This method is useful when setting or retrieving displayed text that might be in Unicode, such as text that appears in a text annotation or bookmark.

A character that cannot be converted to the destination encoding is replaced with a space.

Host encoding is a platform-dependent encoding for the host machine. For non-UNIX Roman systems, it is MacRomanEncoding in Mac OS and WinAnsiEncoding in Windows. In UNIX (except HP-UX) Roman systems, it is ISO8859-1 (ISO Latin-1); for HP-UX, it is HP-ROMAN8. See Appendix D in the PDF Reference for descriptions of MacRomanEncoding, WinAnsiEncoding, and PDFDocEncoding.

For non-Roman systems, the host encoding may be a variety of encodings, which are defined by a CMap (character map). See Section 5.6.4 in the PDF Reference for information on CMaps.

Use IG_PDF_get_host_encoding to determine if a system's host encoding is Roman or not.

Return Value:

Error count

IG_PDF_translate_to_pdf

Group: C API Function

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_translate_to_pdf(
   AT_PDF_BOOL bUseUnicode,
   LPCSTR szInHostStr,
   LONG nInHostStrSize,
   LPSTR szOutPDFStr,
   LONG nOutPDFStrSize,
   LPLONG lpnOutPDFStrBytes
   );
```

Parameters:

bUseUnicode	If true, translate the string to Unicode; otherwise use PDFDocEncoding.
szInHostStr	Pointer to the string to translate (may point to the same memory as szOutPDFStr, allowing strings to translate in place).
nInHostStrSize	Number of bytes in szOutPDFStr.
szOutPDFStr	Pointer to the translated string (may point to the same memory as szInHostStr).
nOutPDFStrSize	The length of the szOutPDFStr buffer, in bytes.
lpnOutPDFStrBytes	Number of bytes in the translated string szOutPDFStr.

Description:

Translates a string from host encoding to PDFDocEncoding or Unicode. This method is useful when using text that must be in PDFDocEncoding or Unicode, such as text in a text annotation, bookmark, or article title.

A character that cannot be converted to the destination encoding is replaced with a space.

For example, it converts \n to a space character (\r is present in PDFDocEncoding and is left unchanged).

Host encoding is a platform-dependent encoding for the host machine. For non-UNIX Roman systems, it is MacRomanEncoding in Mac OS and WinAnsiEncoding in Windows. In UNIX (except HP-UX) Roman systems, it is ISO8859-1 (ISO Latin-1); for HP-UX, it is HP-ROMAN8. See Appendix D in the PDF Reference for descriptions of MacRomanEncoding, WinAnsiEncoding, and PDFDocEncoding.

For non-Roman systems, the host encoding may be a variety of encodings, which are defined by a CMap (character map). See Section 5.6.4 in the PDF Reference for a list of predefined CMaps.

Use IG_PDF_get_host_encoding to determine if a system's host encoding is Roman or not.

Return Value:

Error count

AM_PDF_LONG_TO_FIXED

Group: C API Macro

Declaration:

```
AM_PDF_LONG_TO_FIXED(x)
```

Arguments:

X	Long integer value.

Description:

Converts **x** to AT_PDF_FIXED and returns it.

Return Value:

AT_PDF_FIXED value

AM_PDF_FIXED_ROUND_TO_LONG

Group: C API Macro

Declaration:

```
AM_PDF_FIXED_ROUND_TO_LONG (f)
```

Arguments:

f	Fixed value.

Description:

Converts the AT_PDF_FIXED number **f** to an integer, rounding it to the nearest long integer value and returns it.

Return Value:

Long integer value

AM_PDF_FIXED_TRUNC_TO_LONG

Group: C API Macro

Declaration:

```
AM_PDF_FIXED_TRUNC_TO_LONG (f)
```

Arguments:

f	Fixed value.

Description:

Converts the AT_PDF_FIXED number **f** to an integer, truncating it to the next lower long integer value and returns it.

Return Value:

Long integer value

AM_PDF_SHORT_TO_FIXED

Group: C API Macro

Declaration:

```
AM_PDF_SHORT_TO_FIXED(x)
```

Arguments:

x	Short integer value.

Description:

Converts **x** to AT_PDF_FIXED and returns it.

Return Value:

AT_PDF_FIXED value

AM_PDF_FIXED_ROUND_TO_SHORT

Group: C API Macro

Declaration:

```
AM_PDF_FIXED_ROUND_TO_SHORT (f)
```

Arguments:

f	Fixed value.

Description:

Converts the AT_PDF_FIXED number **f** to an integer, rounding it to the nearest short integer value and returns it.

Return Value:

Short integer value

AM_PDF_FIXED_TRUNC_TO_SHORT

Group: C API Macro

Declaration:

```
AM_PDF_FIXED_TRUNC_TO_SHORT (f)
```

Arguments:

f	Fixed value.

Description:

Converts the AT_PDF_FIXED number **f** to an integer, truncating it to the next lower short integer value and returns it.

Return Value:

Short integer value

AM_PDF_DOUBLE_TO_FIXED

Group: C API Macro

Declaration:

```
AM_PDF_DOUBLE_TO_FIXED (x)
```

Arguments:

x	Double value.

Description:

Converts **x** to AT_PDF_FIXED and returns it.

Return Value:

AT_PDF_FIXED Value

AM_PDF_FIXED_TO_DOUBLE

Group: C API Macro

Declaration:

```
AM_PDF_FIXED_TO_DOUBLE (f)
```

Arguments:

f	Fixed value.

Description:

Converts the AT_PDF_FIXED number f to double value and returns it.

Return Value:

Double Value

LPFNIG_PDF_AUTHPROC

Group: C API Callback

Declaration:

```
typedef AT_PDF_BOOL (LPACCUAPI LPFNIG_PDF_AUTHPROC)(
   HIG_PDF_DOC hDoc,
   LPVOID clientData
   );
```

Arguments:

hDoc	The PDF document to open.
clientData	User-supplied data that was passed in the call to IG_PDF_register_authproc.

Description:

This callback is used by document open routine. It is called when an encrypted document is being opened, to determine whether or not the user is authorized to open the file.

This callback implements whatever authorization strategy you choose and calls the callbacks of the appropriate security handler (the one that was used to secure the document) to obtain and check authorization data.

The LPFNIG_PDF_AUTHPROC should obtain the authorization data (a password) and call IG_PDF_doc_perm_request(). IG_PDF_doc_perm_request in turn calls the document encryption handler's Authorize function, which returns the permissions that the authorization data enables. IG_PDF_doc_perm_request adds these permissions to those currently allowed, and returns the new set of allowed permissions.

Return Value:

TRUE if the user is authorized to open the document, FALSE otherwise.

LPFNIG_PDF_STREAM_PROC

Group: C API Callback

Declaration:

```
typedef LONG (LPACCUAPI LPFNIG_PDF_STREAM_PROC)(
LPSTR lpData,
UINT nDataLen,
LPVOID clientData );
```

Arguments:

lpData	Buffer into which your procedure must place the number of bytes specified by nDataLen.
nDataLen	Number of bytes to read from the stream and place into data.
clientData	User-supplied data that was specified in the call to **IG_PDF_stream_read_CB_register** or **IG_PDF_stream_write_CB_register**.

Description:

Callback for use by **IG_PDF_stream_read_CB_register** and **IG_PDF_stream_write_CB_register**. This procedure must return the number of bytes specified by nDataLen, obtaining them in any way it wishes.

Return Value:

Returns the number of bytes actually read or written.

LPFNIG_PDF_STREAM_DESTROYPROC

Group: C API Callback

Declaration:

```
typedef void (LPACCUAPI LPFNIG_PDF_STREAM_DESTROYPROC)(
LPVOID clientData );
```

Arguments:

lpData	Buffer into which your procedure must place the number of bytes specified by nDataLen.
nDataLen	Number of bytes to read from the stream and place into data.
clientData	User-supplied data that was specified in the call to **IG_PDF_stream_write_CB_register**.

Description:

Callback for use by **IG_PDF_stream_write_CB_register**. Called at end of stream so you can do clean up and free allocated memory.

Return Value:

Returns the number of bytes actually read or written.

LPFNIG_PDF_SYSFONT_ENUMPROC

Group: C API Callback

Declaration:

```
typedef (LPACCUAPI LPFNIG_PDF_SYSFONT_ENUMPROC)(
HIG_PDF_SYSFONT hSysFont,
LPVOID clientData );
```

Arguments:

hSysFont	The system font
clientData	User-supplied data that was specified in the call to **IG_PDF_sysfont_enumerate**.

Description:

Callback for **IG_PDF_sysfont_enumerate**. It is called once for each system font.

Return Value:

Returns true to continue enumeration, false to halt enumeration

LPFNIG_PDE_CLIP_ENUMPROC

Group: C API Callback

Declaration:

```
typedef AT_PDF_BOOL (LPACCUAPI LPFNIG_PDE_CLIP_ENUMPROC)(
HIG_PDE_ELEMENT hElement,
LPVOID clientData );
```

Arguments:

hElement	The PDE Element currently being enumerated.
clientData	User-supplied data that was passed in the call to **IG_PDE_clip_enumerate_elements**.

Description:

Callback for `IG_PDE_clip_enumerate_elements()`, which enumerates all of a PDE Clip's PDE Elements in a flattened manner.

Return Value:

If False, enumeration halts. If True, enumeration continues.

Basic Objects

This chapter describes the basic PDF objects and utility functions used throughout the ImageGear PDF API. These objects provide access to the building blocks used to construct PDF documents. Its methods allow applications to manipulate the low-level data in a PDF file, such as strings, numbers, and dictionaries. Adobe PDF supports eight basic platform-independent types of object:

- Boolean values
- Integer and Fixed (real) numbers
- Strings
- Names
- Arrays
- Dictionaries
- Streams
- The null object

These objects may be labeled so that they can be referred to by other objects. A labeled object is called an indirect object. When a direct object is created, the object itself is returned. As a result, a direct object can only be attached to one other Base object at a time; it cannot, for example, be shared by two different dictionaries. When an indirect object is created, something equivalent to a pointer to the object is returned. As a result, an indirect object can be attached to multiple places in a PDF file simultaneously; it can, for example, be shared by two different dictionaries.

PDF documents are trees of these Base objects. Base objects represent document components such as bookmarks, pages, and fonts. Unlike using the other ImageGear PDF Obejcts methods, using Base Object methods improperlycould result in an invalid PDF file. Therefore, you should not use Base Object methods unless necessary, for example to add private data to portions of a PDF file that cannot be accessed in other ways.

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

Basic Objects

HIG_PDF_BASOBJ	Basic Object – basic PDF object interface. PDF supports eight basic types of object: ■ Boolean values ■ Integer and Fixed (real) numbers ■ Strings ■ Names ■ Arrays ■ Dictionaries ■ Streams ■ The null object Objects may be labeled so that they can be referred to by other objects. A labeled object is called an indirect object.
HIG_PDF_BASBOOL	Basic Boolean – PDF provides boolean objects identified by the keywords true and false. Boolean objects can be used as the values of array elements and dictionary entries.
HIG_PDF_BASINT	Basic Integer – Integer objects represent mathematical integers within a certain interval centered at 0.
HIG_PDF_BASFIXED	Basic Fixed – Fixed objects approximate mathematical real numbers, but with limited range and precision; they are typically represented in fixed-point, rather than floating-point, form.
HIG_PDF_BASSTR	Basic String – A string object consists of a series of bytes—unsigned integer values in the range 0 to 255. The string elements are not integer objects, but are stored in a more compact format.
HIG_PDF_BASNAME	Basic Name – A name object is an atomic symbol uniquely defined by a sequence of characters. Uniquely defined means that any two name objects made up of the same sequence of characters are identically the same object. Atomic means that a name has no internal structure; although it is defined by a sequence of characters, those characters are not “elements” of the name.
HIG_PDF_BASARR	Basic Array – an array object is a one-dimensional collection of objects arranged sequentially. Unlike arrays in many other computer languages, PDF arrays may be heterogeneous; that is, an array’s elements may be any combination of numbers, strings, dictionaries, or any other objects, including other arrays.
HIG_PDF_BASDICT	Basic Dictionary – a dictionary object is an associative table containing pairs of objects, known as the dictionary’s entries. The first element of each entry is the key and the second element is the value. The key must be a name. The value can be any kind of object, including another dictionary.
HIG_PDF_BASNULL	Basic Null – The null object has a type and value that are unequal to those of any other object. There is only one object of type null, denoted by the keyword **null**.

HIG_PDF_BASOBJ

Type: Basic Object

Description:

Handle to the basic object.

Class Members:

IG_PDF_basobj_get_type	Gets an object's type.
IG_PDF_basobj_release	Releases PDF Object.
IG_PDF_basobj_remove	Removes basic object.

IG_PDF_basobj_get_type

Group: Basic Object

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basobj_get_type (HIG_PDF_BASOBJ
hObject,LPLONG lpnType );
```

Arguments:

hObject	Basic object.
lpnType	The object's type.

Description:

Gets an object's type.

Return Value:

Error count.

IG_PDF_basobj_release

Group: Basic Object

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basobj_release (HIG_PDF_BASOBJ
hObject );
```

Arguments:

hObject	Object to release.

Description:

Releases PDF Object.

Return Value:

Error count.

IG_PDF_basobj_remove

Group: Basic Object

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basobj_remove(
   HIG_PDF_BASOBJ hObject
   );
```

Arguments:

hObject	The object to remove.

Description:

Removes basic object. If a composite object (array, dictionary or stream) is removed all the direct objects in it are automatically removed, but the indirect objects in it are not removed.

Return Value:

Error count.

HIG_PDF_BASBOOL

Type: Basic Boolean

Description:

Handle to the basic Boolean object. PDF provides boolean objects identified by the keywords true and false. Boolean objects can be used as the values of array elements and dictionary entries.

Class Members:

IG_PDF_basbool_create	Creates a new boolean object associated with the specified document and having the specified value.
IG_PDF_basbool_get_value	Gets the value of the specified boolean object.

IG_PDF_basbool_create

Group: Basic Boolean

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basbool_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
AT_PDF_BOOL bValue,
LPHIG_PDF_BASOBJ lphBool );
```

Arguments:

hDoc	The document in which the boolean is used.
bIndirect	If True, creates the boolean object as an indirect object. If False, creates the boolean as a direct object.
bValue	The value the new boolean will have.
lphBool	A boolean object.

Description:

Creates a new boolean object associated with the specified document and having the specified value.

Return Value:

Error count.

IG_PDF_basbool_get_value

Group: Basic Boolean

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basbool_get_value (
HIG_PDF_BASOBJ hBool,
LPAT_PDF_BOOL lpbValue );
```

Arguments:

hBool	Object
lpbValue	Value

Description:

Gets the value of the specified boolean object.

Return Value:

Error count.

HIG_PDF_BASINT

Type: Basic Integer Object

Description:

Handle to the basic integer object. Integer objects represent mathematical integers within a certain interval centered at 0.

Class Members:

IG_PDF_basint_create	Creates a new integer object associated with the specified document and having the specified value.
IG_PDF_basint_get_value	Gets the value of the specified integer object.

IG_PDF_basint_create

Group: Basic Integer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basint_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
LONG nValue,
LPHIG_PDF_BASOBJ lphInt );
```

Arguments:

hDoc	The document in which the object is used.
bIndirect	If True, creates the integer as an indirect object. If False, creates the integer as a direct object.
nValue	The value the new integer will have.
lphInt	An integer object.

Description:

Creates a new integer object associated with the specified document and having the specified value.

Return Value:

Error count.

IG_PDF_basint_get_value

Group: Basic Integer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basint_get_value (
HIG_PDF_BASOBJ hInt,
LPLONG lpnValue );
```

Arguments:

hInt	Object
lpnValue	Value

Description:

Gets the value of the specified integer object.

Return Value:

Error count.

HIG_PDF_BASFIXED

Type: Basic Fixed Object

Description:

Handle to the basic fixed object. Fixed objects approximate mathematical real numbers, but with limited range and precision; they are typically represented in fixed-point, rather than floating-point, form.

Class Members:

IG_PDF_basfixed_create	Creates a new fixed object associated with the specified document and having the specified value.
IG_PDF_basfixed_get_value	Gets the value of the specified fixed object.

IG_PDF_basfixed_create

Group: Basic Fixed

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basfixed_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
LONG nValue,
LPHIG_PDF_BASOBJ lphFixed );
```

Arguments:

hDoc	The document in which the object is used.
bIndirect	If True, creates the fixed as an indirect object. If False, creates the fixed as a direct object.
nValue	The value the new fixed will have.
lphFixed	The value the new fixed will have.

Description:

Creates a new fixed object associated with the specified document and having the specified value.

Return Value:

Error count.

IG_PDF_basfixed_get_value

Group: Basic Fixed

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basfixed_get_value (
HIG_PDF_BASOBJ hFixed,
LPLONG lpnValue );
```

Arguments:

hFixed	Object
lpnValue	Value

Description:

Gets the value of the specified fixed object.

Return Value:

Error count.

HIG_PDF_BASSTR

Type: Basic String Object

Description:

Handle to the basic string object. A string object consists of a series of bytes—unsigned integer values in the range 0 to 255. The string elements are not integer objects, but are stored in a more compact format.

Class Members:

IG_PDF_basstr_create	Creates a new string object associated with the specified document and having the specified value.
IG_PDF_basstr_get_value	Copies at most `nLen` bytes from obj's string value into `lpString`, and stores the actual length of the basic string in `lpnBytes`.

IG_PDF_basstr_create

Group: Basic String

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basstr_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
LPBYTE lpString,
LONG nBytes,
LPHIG_PDF_BASOBJ lphString );
```

Arguments:

hDoc	The document in which the object is used.
bIndirect	If True, creates the string as an indirect object. If False, creates the string as a direct object.
lpString	The value the new string will have.
nBytes	The length of lpString.
lphString	The value the new string will have.

Description:

Creates a new string object associated with the specified document and having the specified value.

Return Value:

Error count.

IG_PDF_basstr_get_value

Group: Basic String

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basstr_get_value (
HIG_PDF_BASOBJ hString,
LPBYTE lpString,
LONG nLen,
LPLONG lpnBytes );
```

Arguments:

hString	Object
lpString	The buffer into which the original string value is copied or NULL.
nLen	The length of buffer or 0.
lpnBytes	The length of the original string in bytes. Must be a non-NULL pointer.

Description:

Copies at most `nLen` bytes from obj's string value into `lpString`, and stores the actual length of the basic string in `lpnBytes`.

Return Value:

Error count.

HIG_PDF_BASNAME

Type: Basic Name Object

Description:

Handle to the basic name object. A name object is an atomic symbol uniquely defined by a sequence of characters.

Uniquely defined means that any two name objects made up of the same sequence of characters are identically the same object. Atomic means that a name has no internal structure; although it is defined by a sequence of characters, those characters are not “elements” of the name.

Class Members:

IG_PDF_basname_create	Creates a new name object associated with the specified document and having the specified value.
IG_PDF_basname_get_value	Gets the value of the specified name object.

IG_PDF_basname_create

Group: Basic Name

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basname_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
HIG_PDF_ATOM hNameVal,
LPHIG_PDF_BASOBJ lphName);
```

Arguments:

hDoc	The document in which the object is used.
bIndirect	If True, creates the name as an indirect object. If False, creates the name as a direct object.
hNameVal	The value the new name will have.
lphName	A name object.

Description:

Creates a new name object associated with the specified document and having the specified value.

Return Value:

Error count.

IG_PDF_basname_get_value

Group: Basic Name

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basname_get_value (
HIG_PDF_BASOBJ hName,
LPHIG_PDF_ATOM lphNameVal );
```

Arguments:

hName	Object
lpnNameVal	Value

Description:

Gets the value of the specified name object.

Return Value:

Error count.

HIG_PDF_BASARR

Type: Basic Array Object

Description:

General HIG_PDF_BASOBJ is used to handle to the basic array object. An array object is a one-dimensional collection of objects arranged sequentially. Unlike arrays in many other computer languages, PDF arrays may be heterogeneous; that is, an array's elements may be any combination of numbers, strings, dictionaries, or any other objects, including other arrays.

Class Members:

IG_PDF_basarr_create	Creates a new array of objects.
IG_PDF_basarr_get_length	Gets the number of elements in hArray.
IG_PDF_basarr_get	Gets the specified element from an array.
IG_PDF_basarr_put	Puts the specified object into the specified location in an array.
IG_PDF_basarr_put_int	Puts the specified fixed value into the specified location in an array.
IG_PDF_basarr_put_fixed	Puts the specified fixed value into the specified location in an array.
IG_PDF_basarr_put_bool	Puts the specified boolean value into the specified location in an array.
IG_PDF_basarr_put_name	Puts the specified name value into the specified location in an array.
IG_PDF_basarr_remove	Finds the first element, if any, equal to the specified object and removes it from the array.
IG_PDF_basarr_remove_nth	Checks whether the position is within the array bounds and then removes it from the array and moves each subsequent element to the slot with the next smaller index and decrements the array's length by 1.

IG_PDF_basarr_create

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
UINT nElements,
```

```
LPHIG_PDF_BASOBJ lphArray );
```

Arguments:

hDoc	The document in which the array is used.
bIndirect	If true, creates the array as an indirect object. If false, creates the dictionary as a direct object.
nElements	Number of entries in the array. This value is only a hint — the arrays grow dynamically as needed.
lphArray	New array

Description:

Creates a new array of objects.

Return Value:

Error count.

IG_PDF_basarr_get_length

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_get_length(
HIG_PDF_BASOBJ hArray,
LPLONG lpnLength );
```

Arguments:

hArray	The array.
lpnLength	The number of elements in hArray.

Description:

Gets the number of elements in `hArray`.

Return Value:

Error count.

IG_PDF_basarr_get

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_get(
HIG_PDF_BASOBJ hArray,
UINT nIndex,
LPHIG_PDF_BASOBJ lphObject);
```

Arguments:

hArray	The array from which an element is obtained.
nIndex	The array element to obtain. The first element in an array has an index of zero.
lphObject	The basic object occupying the `nIndex` element of array. Returns `IG_PDF_BASIC_NULL` object if index is outside the array bounds.

Description:

Gets the specified element from an array.

Return Value:

Error count.

IG_PDF_basarr_put

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put(
HIG_PDF_BASOBJ hArray,
UINT nIndex,
HIG_PDF_BASOBJ hObject);
```

Arguments:

hArray	The array in which `hObject` is stored.
nIndex	The location in array to store `hObject`. The first element of an array has an index of zero.
hObject	The object to insert into `hArray`.

Description:

Puts the specified object into the specified location in an array. The array is extended as much as necessary.

Return Value:

Error count.

IG_PDF_basarr_put_int

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_int(
HIG_PDF_BASOBJ hArray,
UINT nIndex,
AT_PDF_BOOL bIndirect,
INT nValue );
```

Arguments:

hArray	The array in which a value is stored.
nIndex	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
nValue	The integer value to insert into `hArray`.

Description:

Puts the specified integer value into the specified location in an array. The array is extended as much as necessary.

Return Value:

Error count.

IG_PDF_basarr_put_fixed

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_fixed(
HIG_PDF_BASOBJ hArray,
UINT nIndex,
AT_PDF_BOOL bIndirect,
AT_PDF_FIXED nValue );
```

Arguments:

hArray	The array in which a value is stored.
nIndex	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
nValue	The fixed value to insert into `hArray`.

Description:

Puts the specified fixed value into the specified location in an array. The array is extended as much as necessary.

Return Value:

Error count.

IG_PDF_basarr_put_bool

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_bool(
HIG_PDF_BASOBJ hArray,
UINT nIndex,
AT_PDF_BOOL bIndirect,
AT_PDF_BOOL bValue );
```

Arguments:

hArray	The array in which a value is stored.
nIndex	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
bValue	The boolean value to insert into `hArray`.

Description:

Puts the specified boolean value into the specified location in an array. The array is extended as much as necessary.

Return Value:

Error count.

IG_PDF_basarr_put_name

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_name(
HIG_PDF_BASOBJ hArray,
UINT nIndex,
AT_PDF_BOOL bIndirect,
HIG_PDF_ATOM nName);
```

Arguments:

hArray	The array in which a value is stored.
nIndex	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
nName	The name value to insert into `hArray`.

Description:

Puts the specified name value into the specified location in an array. The array is extended as much as necessary.

Return Value:

Error count.

IG_PDF_basarr_remove

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_remove(
HIG_PDF_BASOBJ hArray,
HIG_PDF_BASOBJ hObject);
```

Arguments:

hArray	The array in which `hObject` is removed.
hObject	The object to remove.

Description:

Finds the first element, if any, equal to the specified object and removes it from the array.

Return Value:

Error count.

IG_PDF_basarr_remove_nth

Group: Basic Array

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_remove_nth(
HIG_PDF_BASOBJ hArray,
UINT nIndex);
```

Arguments:

hArray	The array to remove the member from.
nIndex	The index for the array member to remove. Array indices start at 0.

Description:

Checks whether the position is within the array bounds and then removes it from the array and moves each subsequent element to the slot with the next smaller index and decrements the array's length by 1.

Return Value:

Error count.

HIG_PDF_BASDICT

Type: Basic Dictionary Object

Description:

General HIG_PDF_BASOBJ is used to handle to the basic dictionary object. A dictionary object is an associative table containing pairs of objects, known as the dictionary's entries. The first element of each entry is the key and the second element is the value. The key must be a name. The value can be any kind of object, including another dictionary.

Class Members:

IG_PDF_basdict_create	Creates a new dictionary.
IG_PDF_basdict_known	Tests whether a specific key is found in the specified dictionary.
IG_PDF_basdict_get	Gets the value of the specified key in the specified dictionary.
IG_PDF_basdict_put	Sets the value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_int	Sets the integer value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_fixed	Sets the boolean value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_bool	Sets the boolean value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_name	Sets the name value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_remove	Removes a key–value pair from a dictionary.

IG_PDF_basdict_create

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_create(
HIG_PDF_DOC hDoc,
AT_PDF_BOOL bIndirect,
UINT nEntries,
LPHIG_PDF_BASOBJ lphDictionary );
```

Arguments:

hDoc	The document in which the dictionary is used.
bIndirect	If true, creates the dictionary as an indirect object. If false, creates the dictionary as a direct object.
nEntries	Number of entries in the dictionary. This value is only a hint — the dictionaries grow dynamically as needed.
lphDictionary	New dictionary

Description:

Creates a new dictionary.

See the PDF Reference for information on dictionary objects that are part of standard PDF, such as annotations or page objects.

Return Value:

Error count.

IG_PDF_basdict_known

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_known(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
LPAT_PDF_BOOL lpbKnown );
```

Arguments:

hDictionary	The dictionary in which to look for key.
hKey	The key to find. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
lpbKnown	True if the value of a key is known (exists and is not null) in `hDictionary`; false otherwise.

Description:

Tests whether a specific key is found in the specified dictionary.

Return Value:

Error count.

IG_PDF_basdict_get

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_get(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
LPHIG_PDF_BASOBJ lphObject);
```

Arguments:

hDictionary	The dictionary or stream from which a value is obtained.
hKey	The key whose value is obtained. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
lphObject	The object associated with the specified key. If key is not present or if its value is null, returns an object of type `IG_PDF_BASIC_NULL`.

Description:

Gets the value of the specified key in the specified dictionary. If called with a stream object instead of a dictionary object, this method gets the value of the specified key from the stream's attributes dictionary.

Return Value:

Error count.

IG_PDF_basdict_put

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
HIG_PDF_BASOBJ hObject);
```

Arguments:

hDictionary	The dictionary in which a value is set.
hKey	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
hObject	The value to set.

Description:

Sets the value of a dictionary key, adding the key to the dictionary if it is not already present.

Return Value:

Error count.

IG_PDF_basdict_put_int

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_int(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
AT_PDF_BOOL bIndirect,
INT nValue );
```

Arguments:

hDictionary	The dictionary in which a value is set.
hKey	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
nValue	The integer value to set.

Description:

Sets the integer value of a dictionary key, adding the key to the dictionary if it is not already present.

Return Value:

Error count.

IG_PDF_basdict_put_fixed

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_fixed(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
AT_PDF_BOOL bIndirect,
AT_PDF_FIXED nValue );
```

Arguments:

hDictionary	The dictionary in which a value is set.
hKey	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
nValue	The fixed value to set.

Description:

Sets the fixed value of a dictionary key, adding the key to the dictionary if it is not already present.

Return Value:

Error count.

IG_PDF_basdict_put_bool

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_bool(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
AT_PDF_BOOL bIndirect,
AT_PDF_BOOL bValue );
```

Arguments:

hDictionary	The dictionary in which a value is set.
hKey	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
bValue	The boolean value to set.

Description:

Sets the boolean value of a dictionary key, adding the key to the dictionary if it is not already present.

Return Value:

Error count.

IG_PDF_basdict_put_name

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_name(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey,
AT_PDF_BOOL bIndirect,
HIG_PDF_ATOM nName);
```

Arguments:

hDictionary	The dictionary in which a value is set.
hKey	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	If true, creates the key value as an indirect object. If false, creates the key value as a direct object.
nName	The name value to set.

Description:

Sets the name value of a dictionary key, adding the key to the dictionary if it is not already present.

Return Value:

Error count.

IG_PDF_basdict_remove

Group: Basic Dictionary

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_remove(
HIG_PDF_BASOBJ hDictionary,
HIG_PDF_ATOM hKey);
```

Arguments:

hDictionary	The dictionary in which a key is removed.
hKey	The key to remove.

Description:

Removes a key–value pair from a dictionary.

Return Value:

Error count.

HIG_PDF_BASNULL

Type: Basic Null Object

Description:

Handle to the basic null object. The null object has a type and value that are unequal to those of any other object. There is only one object of type null, denoted by the keyword *null*.

Class Members:

IG_PDF_basnull_create	Creates a direct **null** object.

IG_PDF_basnull_create

Group: Basic Null

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basnull_create(
   LPHIG_PDF_BASOBJ lphNull );
```

Arguments:

lphNull	Null object.

Description:

Creates a direct **null** object.

Return Value:

Error count.

General Objects

This chapter describes a group of objects that provide access to PDF document's components such as metadata, pages, fonts etc. These objects and methods allow applications to manipulate the PDF content and data. Some of the objects allow you to work with host system fonts and encodings, and the supplementary objects such as Atom and Stream used to simplify and optimize working with PDF content.

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

General Objects	
HIG_PDF_ACTION	Handle to a PDF action object, which is a task that performs when a user clicks on a link or a bookmark.
HIG_PDF_ATOM	Atom - a hashed token used in place of strings to optimize performance (it is much faster to compare Atoms than strings). Many methods use Atoms.
HIG_PDF_BOOKMARK	Handle to a PDF bookmark object, which allows the user to navigate interactively from one part of the document to another.
HIG_PDF_DESTINATION	Handle to a PDF destination object, which represents a particular view of a page in a document.
HIG_PDF_DOC	Document - the underlying PDF representation of a document. Through PDF Document, your application can perform most of the Edit Pages operations (delete, replace, and so on). Thumbnails can be created and deleted through this object. You can set and retrieve document information fields through this object as well.
HIG_PDF_PAGE	Page - a single page in the PDF representation of a document. A page contains a series of objects representing the objects drawn on the page (Graphic), a list of resources used in drawing the page, annotations (Annotation), an optional thumbnail image of the page, and the beads used in any articles that occur on the page.
HIG_PDF_STREAM	Stream - a data stream that may be a buffer in memory, or an arbitrary user-written procedure. Typically used to extract or provide data.
HIG_PDF_STYLE	Style - provides access to information about the fonts, font sizes, and colors used in a Word.
HIG_PDF_SYSENCODING	SysEncoding – provides system encoding for a PDF file.
HIG_PDF_SYSFONT	SysFont – a reference to a font installed in the host system. SysFont methods allow you to list the fonts available in the host system and to find a font in the system that matches a PDE Font, if it is present.
HIG_PDF_WORD	Word - a word in a PDF file. Each word contains a sequence of characters in one or more styles (see Style).
HIG_PDF_WORDFINDER	WordFinder – extracts words from a PDF file, and enumerates the words on a single page or on all pages in a document..

HIG_PDF_ACTION

Type: Atom Object

Description:

Handle to a PDF action object, which is a task that performs when a user clicks on a link or a bookmark. Action types include:

- Going to another view within the same document
- Going to a specified view in another PDF file
- Launching an arbitrary file
- Resolving a URL

See Section 8.5 in the PDF Reference for more information on actions.

Class Members:

IG_PDF_action_create	Creates a new action object.
IG_PDF_action_create_destination	Creates a new action that takes the user to the specified destination view.
IG_PDF_action_create_filename	Creates an action of the specified type from a file name.
IG_PDF_action_delete	Deletes an action object.
IG_PDF_action_get_destination	Gets an action's destination view.
IG_PDF_action_get_dictionary	Gets the dictionary corresponding to an action.
IG_PDF_action_get_filename	Gets a file name from an action.
IG_PDF_action_get_type	Gets an action's type.

IG_PDF_action_create

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_create(
   HIG_PDF_DOC hDoc,
   HIG_PDF_ATOM hType,
   LPHIG_PDF_ACTION lphAction
   );
```

Arguments:

hDoc	The document in which the action is created and used.
hType	The atom corresponding to the action's subtype.
lphAction	The newly created action object.

Description:

Creates a new action object.

Return Value:

Error count.

IG_PDF_action_create_destination

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_create_destination(
    HIG_PDF_DOC hDoc,
    HIG_PDF_DESTINATION hDest,
    LPHIG_PDF_ACTION lphAction
    );
```

Arguments:

hDoc	The document in which the action is created and used.
hDest	The destination.
lphAction	The newly created action object.

Description:

Creates a new action that takes the user to the specified destination view. This method can only be used for destinations in the same document as the source document. Cross-document links must be built up from the base level, populating the Action dictionary for the GotoR action as described in Section 8.5.3 in the PDF Reference.

Return Value:

Error count.

IG_PDF_action_create_filename

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_create_filename(
   HIG_PDF_DOC hDoc,
   HIG_PDF_ATOM hType,
   LPCSTR szFileName,
   LPHIG_PDF_ACTION lphAction
   );
```

Arguments:

hDoc	The document in which the action is created and used.
hType	The type of action to create.
szFileName	The file name.
lphAction	The newly created action object.

Description:

Creates an action of the specified type from a file name.

Return Value:

Error count.

IG_PDF_action_delete

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_delete(
   HIG_PDF_ACTION hAction
   );
```

Arguments:

hAction	The action object to delete.

Description:

Deletes an action object.

Return Value:

Error count.

IG_PDF_action_get_destination

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_destination(
   HIG_PDF_ACTION hAction,
   LPHIG_PDF_DESTINATION lphDest
   );
```

Arguments:

hAction	The action whose destination is obtained.
lphDest	The action's destination, which may be either an explicit or named (basic string or name object). Use the `IG_PDF_destination_resolve` on this returned value to obtain an explicit destination.

Description:

Gets an action's destination view. This only works for actions that contain a view destination - that is, actions whose type is GoTo. For named destinations, this method may return a basic string or name object. See Section 8.2.1 in the PDF Reference for more information on named destinations.

NOTE: Since this method may not return an explicit destination, use the `IG_PDF_destination_resolve` on the returned value to obtain an explicit destination.

Return Value:

Error count.

IG_PDF_action_get_dictionary

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_dictionary(
   HIG_PDF_ACTION hAction,
   LPHIG_PDF_BASOBJ lphDictionary
```

```
);
```

Arguments:

hAction	The action whose dictionary is obtained.
lphDictionary	Dictionary object for the action.

Description:

Gets the dictionary corresponding to an action.

Return Value:

Error count.

IG_PDF_action_get_filename

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_filename(
   HIG_PDF_ACTION hAction,
   LPSTR lpBuf,
   AT_INT nSize,
   LPAT_INT lpnLen
   );
```

Arguments:

hAction	The action whose file name is obtained.
lpBuf	The buffer to return a file name.
nSize	Size of lpBuf.
lpnLen	Length of the file name.

Description:

Gets a file name from an action. Not all types of actions have file names; this method only works for actions that contain a file specification. See Section 8.5 in the PDF Reference for more information on the contents of various types of actions.

Return Value:

Error count.

IG_PDF_action_get_type

Group: Action

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_type(
   HIG_PDF_ACTION hAction,
   LPHIG_PDF_ATOM lphType
   );
```

Arguments:

hAction	The action whose type is obtained.
lphType	The atom corresponding to the action's type.

Description:

Gets an action's type.

Return Value:

Error count.

HIG_PDF_ATOM

Type: Atom Object

Description:

Handle to the PDF atom object. A hashed token used in place of strings to optimize performance (it is much faster to compare Atoms than strings). Many methods use Atoms.

Class Members:

IG_PDF_atom_from_string	Gets the Atom for the specified string.
IG_PDF_atom_get_string	Gets the string associated with the specified Atom.

IG_PDF_atom_from_string

Group: Atom

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_atom_from_string(
LPCSTR lpString,
LPHIG_PDF_ATOM lphAtom);
```

Arguments:

lpString	The string for which an atom is obtained.
lphAtom	Atom return value

Description:

Gets the Atom for the specified string. You can also use this method to create an Atom, since it creates one for the string if one does not already exist.

If an Atom already exists for `lpString`, the existing Atom is returned. Thus Atoms may be compared for equality of the underlying string.

Because Atoms cannot be deleted, they are useful for strings that are used many times, but are not advisable for strings that have a short lifetime. For the same reason, it is not a good idea to create large numbers of Atoms.

Return Value:

Error count.

IG_PDF_atom_get_string

Group: Atom

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_atom_get_string(
HIG_PDF_ATOM hAtom,
LPCSTR *lpString
);
```

Arguments:

hAtom	The Atom whose string is obtained.
lpString	The string corresponding to `hAtom`. Returns an empty string if `hAtom` is equal to IG_PDF_ATOM_NULL or NULL if the `hAtom` has not been defined.

Description:

Gets the string associated with the specified Atom.

Return Value:

Error count.

HIG_PDF_BOOKMARK

Type: Object

Description:

Handle to a PDF bookmark object, which allows the user to navigate interactively from one part of the document to another. It consists of a tree-structured hierarchy of bookmarks. Each bookmark has:

- A title that appears on screen
- An action that specifies what happens when the user clicks on the bookmark

The typical action for a user-created bookmark is to move to another location in the current document or outside of it. Each bookmark in the bookmark tree structure has zero or more children that appear indented on screen, and zero or more siblings that appear at the same indentation level. All bookmarks except the bookmark at the top level of the hierarchy have an one parent, i.e. the bookmark under which it is indented. A bookmark is said to be open, if its children are visible on the screen, and closed, if they are not.

See the section 8.2.2, “Document Outline,” in the PDF Reference for more information on bookmarks.

Class Members:

IG_PDF_bookmark_add_child	Adds `hChildBookmark` as the last child of parent.
IG_PDF_bookmark_add_new_child	Adds a new bookmark to the tree containing hBookmark.
IG_PDF_bookmark_add_next	Adds hNewNext as the new right sibling to hBookmark.
IG_PDF_bookmark_add_new_sibling	Adds a new bookmark to the tree containing hBookmark as the new right sibling.
IG_PDF_bookmark_add_prev	Adds `hNewPrev` as the new left sibling to `hBookmark`.
IG_PDF_bookmark_add_subtree	Adds a copy of the bookmark sub-tree source to `hBookmark`.
IG_PDF_bookmark_delete	Deletes a bookmark object.
IG_PDF_bookmark_find_title	Gets the first bookmark whose title is `lpTitle`.
IG_PDF_bookmark_get_action	Gets hBookmark's action.
IG_PDF_bookmark_get_color	Gets the color of the specified bookmark.
IG_PDF_bookmark_get_count	Gets the number of open bookmarks in a sub-tree.
IG_PDF_bookmark_get_first_child	Gets hBookmark's first child.
IG_PDF_bookmark_get_flags	Gets the flags of the specified bookmark.
IG_PDF_bookmark_get_indent	Gets the indentation level of a bookmark in its containing tree.
IG_PDF_bookmark_get_last_child	Gets hBookmark's last child.
IG_PDF_bookmark_get_next	Gets hBookmark's next (right) sibling.
IG_PDF_bookmark_get_parent	Gets hBookmark's parent bookmark.
IG_PDF_bookmark_get_prev	Gets hBookmark's previous (left) sibling.
IG_PDF_bookmark_get_title	Gets hBookmark's title.
IG_PDF_bookmark_has_children	Tests whether a bookmark has children.
IG_PDF_bookmark_is_open	Tests whether a bookmark is open.
IG_PDF_bookmark_remove	Removes `hBookmark` sub-tree from the bookmark tree containing it.
IG_PDF_bookmark_remove_action	Removes hBookmark's action.
IG_PDF_bookmark_set_action	Sets hBookmark 's action.

IG_PDF_bookmark_set_color	Sets hBookmark 's color.
IG_PDF_bookmark_set_flags	Sets the flags of the specified bookmark.
IG_PDF_bookmark_set_open	Opens or closes a bookmark.
IG_PDF_bookmark_set_title	Sets hBookmark's title.
IG_PDF_bookmark_unlink	Unlinks a bookmark from the bookmark tree that contains it.

IG_PDF_bookmark_add_child

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_child(
    HIG_PDF_BOOKMARK hBookmark,
    HIG_PDF_BOOKMARK hChildBookmark
    );
```

Arguments:

hBookmark	The parent of the bookmark being added.
hChildBookmark	The bookmark that will become the last child of hBookmark. `hChildBookmark` must have been previously unlinked.

Description:

Adds `hChildBookmark` as the last child of parent, adjusting the tree containing parent appropriately. If parent previously had no children, it is open after the child is added.

Return Value:

Error count.

IG_PDF_bookmark_add_new_child

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_new_child(
    HIG_PDF_BOOKMARK hBookmark,
    LPSTR lpszInitialText,
    LPHIG_PDF_BOOKMARK lphChildBookmark
    );
```

Arguments:

hBookmark	The bookmark to which a new last child is added.
lpszInitialText	The new bookmark's title.
lphChildBookmark	The newly created bookmark.

Description:

Adds a new bookmark to the tree containing `hBookmark`, as the new last child of `hBookmark`. If `hBookmark` previously had no children, it will be open after the child is added.

Return Value:

Error count.

IG_PDF_bookmark_add_next

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_next(
   HIG_PDF_BOOKMARK hBookmark,
   HIG_PDF_BOOKMARK hNewNext
   );
```

Arguments:

hBookmark	The bookmark that will receive a new right sibling.
hNewNext	The bookmark to become the new right sibling of hBookmark. hNewNext must have been previously unlinked.

Description:

Adds `hNewNext` as the new right sibling to `hBookmark`.

Return Value:

Error count.

IG_PDF_bookmark_add_new_sibling

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_new_sibling(
```

```
    HIG_PDF_BOOKMARK hBookmark,
    LPSTR lpszInitialText,
    LPHIG_PDF_BOOKMARK lphSiblingBookmark
    );
```

Arguments:

hBookmark	The bookmark that will be the left sibling of the new bookmark.
lpszInitialText	The new bookmark's title.
lphSiblingBookmark	The newly created bookmark.

Description:

Adds a new bookmark to the tree containing `hBookmark` as the new right sibling.

Return Value:

Error count.

IG_PDF_bookmark_add_prev

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_prev(
   HIG_PDF_BOOKMARK hBookmark,
   HIG_PDF_BOOKMARK hNewPrev
   );
```

Arguments:

hBookmark	The bookmark that will receive a new left sibling.
hNewPrev	The bookmark to become the new left sibling of hBookmark. hNewPrev must have been previously unlinked.

Description:

Adds `hNewPrev` as the new left sibling to `hBookmark`, adjusting the tree containing `hBookmark` appropriately.

Return Value:

Error count.

IG_PDF_bookmark_add_subtree

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_subtree(
   HIG_PDF_BOOKMARK hBookmark,
   HIG_PDF_BOOKMARK hSubtree,
   LPSTR lpszSourceTitle
   );
```

Arguments:

hBookmark	The bookmark to which the sub-tree source is added as a new last child.
hSubtree	The bookmark sub-tree to add.
lpszSourceTitle	The new bookmark's title.

Description:

Adds a copy of the bookmark sub-tree source to `hBookmark`, as a new last child of `hBookmark`. This

new item will have the text value `lpszSourceTitle`, will be open, and will have no destination attribute. `hSubtree` must have been previously unlinked. If `hBookmark` previously had no children, it will be open after the sub-tree is added.

Return Value:

Error count.

IG_PDF_bookmark_delete

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_delete(
   HIG_PDF_BOOKMARK hBookmark
   );
```

Arguments:

hBookmark	The bookmark object to delete.

Description:

Deletes a bookmark object.

Return Value:

Error count.

IG_PDF_bookmark_find_title

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_find_title(
   HIG_PDF_BOOKMARK hBookmark,
   LPSTR lpTitle,
   AT_INT nTitleLen,
   AT_INT nMaxDepth,
   LPHIG_PDF_BOOKMARK lphBookmark
   );
```

Arguments:

hBookmark	The root of the bookmark sub-tree to search.
lpTitle	The text value to search for.
nTitleLen	The length of lpTitle.
nMaxDepth	The number of sub-tree levels to search, not counting the root level. ■ 0 - Only look at hBookmark, not at any of its children. ■ 1 - Check hBookmark and its children, but not any grandchildren or great grandchildren, and so on. ■ -1 - Check the entire sub-tree.
lphBookmark	The bookmark with the specified title or NULL if there is no such bookmark.

Description:

Gets the first bookmark whose title is `lpTitle`.

Return Value:

Error count.

IG_PDF_bookmark_get_action

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_action(
   HIG_PDF_BOOKMARK hBookmark,
   LPHIG_PDF_ACTION lphAction
```

```
);
```

Arguments:

hBookmark	The bookmark whose action is obtained.
lphAction	The bookmark's action.

Description:

This method gets hBookmark's action.

Return Value:

Error count.

IG_PDF_bookmark_get_color

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_color(
   HIG_PDF_BOOKMARK hBookmark,
   LPAT_PDF_COLORVALUE lpBookmarkColor
   );
```

Arguments:

hBookmark	The bookmark whose color is obtained.
lpBookmarkColor	Color of the bookmark.

Description:

Gets the color of the specified bookmark.

Return Value:

Error count.

IG_PDF_bookmark_get_count

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_count(
   HIG_PDF_BOOKMARK hBookmark,
   LPAT_INT lpnCount
```

```
);
```

Arguments:

hBookmark	The root bookmark of a sub-tree to count.
lpnCount	Number of open bookmarks in the sub-tree (not including hBookmark).

Description:

Gets the number of open bookmarks in a sub-tree.

Return Value:

Error count.

IG_PDF_bookmark_get_first_child

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_first_child(
   HIG_PDF_BOOKMARK hBookmark,
   LPHIG_PDF_BOOKMARK lphFirstChild
   );
```

Arguments:

hBookmark	The bookmark whose first child is obtained.
lphFirstChild	First child of hBookmark or `NULL`, if `hBookmark` has no children.

Description:

Gets hBookmark's first child.

Return Value:

Error count.

IG_PDF_bookmark_get_flags

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_flags(
   HIG_PDF_BOOKMARK hBookmark,
   LPAT_INT lpnFlags
```

```
);
```

Arguments:

hBookmark	The bookmark whose flags are obtained.
lpnFlags	Bookmark's flags. The OR value of the **enumIGPDFBookmarkFlags**.

Description:

Gets the flags of the specified bookmark.

Return Value:

Error count.

IG_PDF_bookmark_get_indent

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_indent(
   HIG_PDF_BOOKMARK hBookmark,
   LPAT_INT lpnIndent
   );
```

Arguments:

hBookmark	The bookmark whose indentation level is obtained.
lpnIndent	The indentation level of `hBookmark` in its containing tree. The root level has an indentation level of zero.

Description:

Gets the indentation level of a bookmark in its containing tree.

Return Value:

Error count.

IG_PDF_bookmark_get_last_child

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_last_child(
   HIG_PDF_BOOKMARK hBookmark,
```

```
    LPHIG_PDF_BOOKMARK lphLastChild
    );
```

Arguments:

hBookmark	The bookmark whose last child is obtained.
lphLastChild	Last child of hBookmark or `NULL` if `hBookmark` has no children.

Description:

Gets hBookmark's last child.

Return Value:

Error count.

IG_PDF_bookmark_get_next

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_next(
   HIG_PDF_BOOKMARK hBookmark,
   LPHIG_PDF_BOOKMARK lphNext
   );
```

Arguments:

hBookmark	The bookmark whose right sibling is obtained.
lphNext	hBookmark's next (right) sibling or NULL if hBookmark has no next sibling (it is its parent's last child).

Description:

Gets hBookmark's next (right) sibling.

Return Value:

Error count.

IG_PDF_bookmark_get_parent

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_parent(
```

```
    HIG_PDF_BOOKMARK hBookmark,
    LPHIG_PDF_BOOKMARK lphParent
    );
```

Arguments:

hBookmark	The bookmark whose parent is obtained.
lphParent	Parent bookmark of `hBookmark` or NULL, if `hBookmark` is the root of its tree.

Description:

Gets hBookmark's parent bookmark.

Return Value:

Error count.

IG_PDF_bookmark_get_prev

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_prev(
   HIG_PDF_BOOKMARK hBookmark,
   LPHIG_PDF_BOOKMARK lphPrev
   );
```

Arguments:

hBookmark	The bookmark whose left sibling is obtained.
lphPrev	Previous (left) sibling of `hBookmark` or `NULL`, if `hBookmark` has no previous sibling (it is its parent's first child).

Description:

Gets hBookmark's previous (left) sibling.

Return Value:

Error count.

IG_PDF_bookmark_get_title

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_title(
   HIG_PDF_BOOKMARK hBookmark,
   LPSTR szBuffer,
   AT_INT nSize,
   LPAT_INT lpnBytes
   );
```

Arguments:

hBookmark	The bookmark whose title is obtained.
szBuffer	Buffer into which the title will be written. If `szBuffer` is non-NULL, its length is assumed to be `nSize + 1`, because a null byte is appended to the title.
nSize	The size of `szBuffer`.
lpnBytes	The number of bytes copied into `szBuffer`, not counting the trailing null byte. If `szBuffer` is `NULL`, the number of bytes in the bookmark is returned.

Description:

Gets hBookmark's title.

Return Value:

Error count.

IG_PDF_bookmark_has_children

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_has_children(
   HIG_PDF_BOOKMARK hBookmark,
   LPAT_PDF_BOOL lpbHasChildren
   );
```

Arguments:

hBookmark	The bookmark to test.
lpbHasChildren	TRUE if `hBookmark` has any children, FALSE otherwise.

Description:

This function tests whether a bookmark has children or not.

Return Value:

Error count.

IG_PDF_bookmark_is_open

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_is_open(
    HIG_PDF_BOOKMARK hBookmark,
    LPAT_PDF_BOOL lpbOpen
    );
```

Arguments:

hBookmark	The bookmark to test.
lpbOpen	TRUE, if `hBookmark` is open, FALSE otherwise.

Description:

Tests whether a bookmark is open. An open bookmark shows all its children.

Return Value:

Error count.

IG_PDF_bookmark_remove

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_remove(
    HIG_PDF_BOOKMARK hBookmark
    );
```

Arguments:

hBookmark	The root bookmark of the sub-tree to remove.

Description:

Removes `hBookmark` sub-tree from the bookmark tree containing it.

Return Value:

Error count.

IG_PDF_bookmark_remove_action

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_remove_action(
    HIG_PDF_BOOKMARK hBookmark
    );
```

Arguments:

hBookmark	The bookmark whose action is removed.

Description:

Removes hBookmark's action.

Return Value:

Error count.

IG_PDF_bookmark_set_action

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_action(
    HIG_PDF_BOOKMARK hBookmark,
    HIG_PDF_ACTION hAction
    );
```

Arguments:

hBookmark	The bookmark whose action is set.
hAction	The bookmark's action to set.

Description:

Sets hBookmark 's action.

Return Value:

Error count.

IG_PDF_bookmark_set_color

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_color(
    HIG_PDF_BOOKMARK hBookmark,
    LPAT_PDF_COLORVALUE lpBookmarkColor
    );
```

Arguments:

hBookmark	The bookmark whose color is set.
lpBookmarkColor	The bookmark's color to set. Must be in `IG_PDF_DEVICE_RGB`.

Description:

Sets hBookmark 's color.

Return Value:

Error count.

IG_PDF_bookmark_set_flags

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_flags(
    HIG_PDF_BOOKMARK hBookmark,
    AT_INT nFlags
    );
```

Arguments:

hBookmark	The bookmark whose flags are set.
nFlags	Bookmark's flags. The OR value of the **enumIGPDFBookmarkFlags**.

Description:

Sets the flags of the specified bookmark.

Return Value:

Error count.

IG_PDF_bookmark_set_open

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_open(
   HIG_PDF_BOOKMARK hBookmark,
   AT_PDF_BOOL bIsOpen
   );
```

Arguments:

hBookmark	The bookmark to open or close.
bIsOpen	TRUE, if the bookmark is opened, FALSE if the bookmark is closed..

Description:

Opens or closes a bookmark. An open bookmark shows its children, while a closed bookmark does not.

Return Value:

Error count.

IG_PDF_bookmark_set_title

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_title(
   HIG_PDF_BOOKMARK hBookmark,
   LPCSTR lpTitle,
   AT_INT nTitleLen
```

```
);
```

Arguments:

hBookmark	The bookmark whose title is set.
lpTitle	String containing the bookmark's new title.
nTitleLen	The size of `lpTitle`.

Description:

This function sets hBookmark's title.

Return Value:

Error count.

IG_PDF_bookmark_unlink

Group: Bookmark

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_unlink(
   HIG_PDF_BOOKMARK hBookmark
   );
```

Arguments:

hBookmark	The bookmark to unlink.

Description:

Unlinks a bookmark from the bookmark tree that contains it, and adjusts the tree appropriately.

Return Value:

Error count.

HIG_PDF_STREAM

Type: Stream Object

Description:

Handle to the PDF stream object. A data stream that may be a buffer in memory, or an arbitrary user-written procedure. Typically used to extract or provide data.

Class Members:

IG_PDF_stream_open_mem_for_read	Creates a read-only PDF stream from a memory-resident buffer.
IG_PDF_stream_read_CB_register	Creates a read-only PDF stream from an arbitrary data-producing procedure.
IG_PDF_stream_write_CB_register	Creates a PDF stream from an arbitrary data-producing procedure.
IG_PDF_stream_read	Creates a read-only PDF stream from an arbitrary data-producing procedure.
IG_PDF_stream_flush	Flushes any buffered data to the specified stream.
IG_PDF_stream_close	Closes the specified stream.

IG_PDF_stream_open_mem_for_read

Group: Stream

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_open_mem_for_read(
LPSTR lpData,
UINT nLen,
LPHIG_PDF_STREAM lphStream );
```

Arguments:

lpData	Buffer containing the data to read into the stream. This data buffer must not be disposed of until the stream is closed.
nLen	Length of data, in bytes.
lphStream	Handle to the new stream.

Description:

Creates a read-only PDF stream from a memory-resident buffer. The stream is seek-able.

Return Value:

Error count.

IG_PDF_stream_read_CB_register

Group: Stream

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_read_CB_register(
LPFNIG_PDF_STREAM_PROC lpfnReadProc,
LPVOID lpClientData,
LPHIG_PDF_STREAM lphStream );
```

Arguments:

lpfnReadProc	User-supplied callback that supplies the stream's data.
lpClientData	Pointer to user-supplied data to pass to `lpfnReadProc` each time it is called.
lphStream	Handle to the new stream.

Description:

Creates a read-only PDF stream from an arbitrary data-producing procedure. The stream is not seek-able.

`lpfnReadProc` is called when the client of the stream attempts to read data from it.

Return Value:

Error count.

IG_PDF_stream_write_CB_register

Group: Stream

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_write_CB_register(
LPFNIG_PDF_STREAM_PROC lpfnWriteProc,
LPFNIG_PDF_STREAM_DESTROYPROC lpfnDestroyProc,
LPVOID lpClientData,
LPHIG_PDF_STREAM lphStream );
```

Arguments:

lpfnWriteProc	User-supplied callback that provides the data for the stream.
lpfnDestroyProc	User-supplied callback that destroys the specified stream.
lpClientData	Pointer to user-supplied data to pass to `lpfnWriteProc` each time it is called.
lphStream	Handle to the new stream.

Description:

Creates a PDF stream from an arbitrary data-producing procedure. The stream is not seek-able.

Return Value:

Error count.

IG_PDF_stream_read

Group: Stream

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_read(
HIG_PDF_STREAM hStream,
LPSTR lpBuffer,
LONG nItems,
SHORT nItemSize,
LPLONG lpnItemsRead );
```

Arguments:

hStream	The stream from which data is read.
lpBuffer	Buffer into which data is written.
nItems	Number of items to read. The amount of data read into the memory buffer will be `nItems` × `nItemSize`, unless an EOF is encountered first. The relative values of `nItems` and `nItemSize` really do not matter; the only thing that matters is their product. It is often convenient to set `nItemSize` to 1, so that `nItems` is the number of bytes to read.
nItemSize	Number of bytes in an item in the stream.
lpnItemsRead	The number of items (not bytes) read.

Description:

Creates a read-only PDF stream from an arbitrary data-producing procedure. The stream is not seek-able.

lpfnReadProc is called when the client of the stream attempts to read data from it.

Return Value:

Error count.

IG_PDF_stream_flush

Group: Stream

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_flush(
HIG_PDF_STREAM hStream,
LPLONG lpnResult );
```

Arguments:

hStream	The stream to flush.
lpnResult	0 if successful, otherwise non-zero.

Description:

Flushes any buffered data to the specified stream.

Return Value:

Error count.

IG_PDF_stream_close

Group: Stream

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_close(
HIG_PDF_STREAM hStream );
```

Arguments:

hStream	The stream to close.

Description:

Closes the specified stream.

Return Value:

Error count.

HIG_PDF_DESTINATION

Type: Destination Object

Description:

Handle to a PDF destination object, which represents a particular view of a page in a document. It contains a reference to a page, a rectangle on that page, and information specifying how to adjust the view to fit the window's size and shape. See section 8.2, "Document-Level Navigation," in the PDF Reference for more information on destinations.

Class Members:

IG_PDF_destination_create	Creates a new destination object.
IG_PDF_destination_delete	Deletes a destination object.
IG_PDF_destination_get_explicit_attrs	Gets a destination's fit type, destination rectangle, and zoom factor.
IG_PDF_destination_get_named_attrs	Gets a destination's named attributes.
IG_PDF_destination_get_type	Gets a destination's type.
IG_PDF_destination_remove	Removes a view destination object.
IG_PDF_destination_resolve	Resolves a destination.

IG_PDF_destination_create

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_create(
   HIG_PDF_DOC hDoc,
   HIG_PDF_PAGE hPage,
   HIG_PDF_ATOM hInitialFitType,
   LPAT_PDF_FIXEDRECT lpInitialRect,
   AT_PDF_FIXED nInitialZoom,
   LPHIG_PDF_DESTINATION lphDest
   );
```

Arguments:

hDoc	The document in which the destination is used.
hPage	The destination page.
hInitialFitType	Destination fit type. Must be one of the View Destination Fit Types.
lpInitialRect	Pointer to a AT_PDF_FIXEDRECT specifying the destination rectangle, specified in user space coordinates. The appropriate information will be extracted from lpInitialRect, depending on hInitialFitType, to create the destination. All four of lpInitialRect's components should be set.
nInitialZoom	The zoom factor to set for the destination. Used only if hInitialFitType is XYZ. Use the predefined value IG_PDF_DEST_NULL to indicate a NULL zoom factor.
lphDest	The newly created destination object.

Description:

Creates a new destination object.

Return Value:

Error count.

IG_PDF_destination_delete

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_delete(
   HIG_PDF_DESTINATION hDest
   );
```

Arguments:

hDest	The destination object to delete.

Description:

Deletes a destination object.

Return Value:

Error count.

IG_PDF_destination_get_explicit_attrs

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_get_explicit_attrs(
   HIG_PDF_DESTINATION hDest,
   LPAT_INT lpnPageNum,
   LPHIG_PDF_ATOM lphFitType,
   LPAT_PDF_FIXEDRECT lpRect,
   LPAT_PDF_FIXED lpnZoom
   );
```

Arguments:

hDest	The destination whose explicit attributes are obtained.
lpnPageNum	The page number of the destination's page..
lphFitType	Destination fit type. One of the Destination Fit Types values.
lpRect	Pointer to a AT_PDF_FIXEDRECT containing the destination's rectangle, specified in user space coordinates.
lpnZoom	The destination's zoom factor.

Description:

Gets a destination's fit type, destination rectangle, and zoom factor. Applies only to `IG_PDF_DEST_EXPLICIT` type of destination.

Return Value:

Error count.

IG_PDF_destination_get_named_attrs

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_get_named_attrs(
   HIG_PDF_DESTINATION hDest,
```

```
    LPHIG_PDF_BASOBJ lphName
    );
```

Arguments:

hDest	The destination whose named attributes are obtained.
lphName	Basic array object for the destination. Returns NULL if the destination is invalid.

Description:

Gets a destination's named attributes. Applies only to `IG_PDF_DEST_NAMED` type of destination.

Return Value:

Error count.

IG_PDF_destination_get_type

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_get_type(
   HIG_PDF_DESTINATION hDest,
   LPAT_INT lpnType
   );
```

Arguments:

hDest	The destination whose type is obtained.
lpnType	The destination type. One of the **`enumIGPDFDestinationType`** values.

Description:

Gets a destination's type.

Return Value:

Error count.

IG_PDF_destination_remove

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_remove(
   HIG_PDF_DESTINATION hDest
   );
```

Arguments:

hDest	The destination to remove.

Description:

Removes a view destination object.

Return Value:

Error count.

IG_PDF_destination_resolve

Group: Destination

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_resolve(
   HIG_PDF_DESTINATION hDest,
   HIG_PDF_DOC hDoc,
   LPHIG_PDF_DESTINATION lphResolvedDest
   );
```

Arguments:

hDest	The destination whose type is obtained.
hDoc	The PDF document that contains the destination.
lphResolvedDest	The resolved view destination.

Description:

Resolves a destination. hDest is the value of the D key in an action. It can be a real destination (an array) or a name. If it is a name, look it up in hDoc's Dests dictionary. The value found there can be a real destination (an array) or a dictionary. If it's a dictionary, look up the D key in that dictionary. This method is useful for getting an explicit view destination from an action.

Return Value:

Error count.

HIG_PDF_DICTIONARY

Type: Dictionary Object

Desciption:

Handle to a PDF dictionary object, which represents an optional-content membership dictionary object.

Class Members:

IG_PDF_dictionary_create	Creates a new optional-content membership dictionary object in the given document for the given layers and visibility policy.
IG_PDF_dictionary_get_layer	Gets the layer with the specified index in a membership dictionary.
IG_PDF_dictionary_get_layer_count	Gets the number of layers listed in a membership dictionary.
IG_PDF_dictionary_get_unique_id	Returns some 32bit integer that is unique for all Dictionary objects.
IG_PDF_dictionary_get_vis_policy	Gets the optional-content membership dictionary's visibility policy.
IG_PDF_dictionary_release	Releases the native object and frees memory.

IG_PDF_dictionary_create

Group: Dictionary

Declaration:

```
IG_PDF_dictionary_create(
   HIG_PDF_DOC hDoc,
   LPHIG_PDF_LAYER hLayers,
   AT_INT nLayersCount,
   AT_INT policy,
   LPHIG_PDF_DICTIONARY lphDictionary)
```

Arguments:

hDoc	The document in which the dictionary is used.
hLayers	Array of layers to be the members of the dictionary.
nLayersCount	The number of layers.
policy	The visibility policy that determines the visibility of content with respect to the ON/OFF state of the layers listed in the dictionary.
lphDictionary	The newly created dictionary object.

Description:

Creates a new optional-content membership dictionary object in the given document for the given layers and visibility policy.

To add layer to the existing dictionary, get the current layers' list, modify it, then create a new dictionary with the new list of layers.

Return Value:

The newly created dictionary object, or NULL if no layers are supplied.

IG_PDF_dictionary_get_layer

Group: Dictionary

Declaration:

```
IG_PDF_dictionary_get_layer(
   HIG_PDF_DICTIONARY hDictionary,
   UINT nIndex,
   LPHIG_PDF_LAYER lpLayer)
```

Arguments:

hDictionary	The membership dictionary whose layer is obtained.
nIndex	The index of the needed layer in the dictionary.
lpLayer	The layer object.

Description:

Gets the layer with the specified index in a membership dictionary.

Return Value:

Layer object.

IG_PDF_dictionary_get_layer_count

Group: Dictionary

Declaration:

```
IG_PDF_dictionary_get_layer_count(
   HIG_PDF_DICTIONARY hDictionary,
   LPUINT lpnCount)
```

Arguments:

hDictionary	The membership dictionary whose layers count is obtained.
lpnCount	The count of the document's layers.

Description:

Gets the number of layers listed in a membership dictionary.

Return Value:

The count of the document's layers.

IG_PDF_dictionary_get_unique_id

Group: Dictionary

Declaration:

```
IG_PDF_dictionary_get_unique_id(
   HIG_PDF_DICTIONARY hDictionary,
   LPUINT lpnUniqueId)
```

Arguments:

hDictionary	Dictionary object.
lpnUniqueId	The unique identifier.

Description:

Returns some 32bit integer that is unique for all Dictionary objects. In other words, it is guaranteed that there cannot be two Dictionary objects with the same UniqueIds.

Can be used for Dictionary objects' identification.

Return Value:

An unique identifier of this Dictionary Object.

IG_PDF_dictionary_get_vis_policy

Group: Dictionary

Declaration:

```
IG_PDF_dictionary_get_vis_policy(
   HIG_PDF_DICTIONARY hDictionary,
   LPAT_INT lpPolicy)
```

Arguments:

hDictionary	The dictionary whose policy is obtained.
lpPolicy	The visibility policy.

Description:

Gets the optional-content membership dictionary's visibility policy, which determines the visibility of content with respect to the ON-OFF state of the layers listed in the dictionary.

Return Value:

The visibility policy.

IG_PDF_dictionary_release

Group: Dictionary

Declaration:

```
IG_PDF_dictionary_release(
   HIG_PDF_DICTIONARY hDictionary)
```

Arguments:

hDictionary	Dictionary object to release.

Description:

Releases the native object and frees memory.

Return Value:

Nothing.

HIG_PDF_DOC

Type: Document Object

Description:

Handle to the PDF document object. The underlying PDF representation of a document. Through PDF Document, your application can perform most of the Edit Pages operations (delete, replace, and so on). Thumbnails can be created and deleted through this object. You can set and retrieve document information fields through this object as well.

Class Members:

IG_PDF_doc_create	This function creates a PDF document and attaches it to HMIGEAR.
IG_PDF_doc_create_new_page	This function creates a new PDF page for the hMPIDoc.
IG_PDF_doc_delete_pages	Deletes the specified pages.
IG_PDF_doc_insert_pages	This function inserts nPageCount pages from hDoc2 into hDoc.
IG_PDF_doc_get_bookmark	Gets the root of the document's bookmark tree.
IG_PDF_doc_get_info	Gets the value of a key in a document's Info dictionary, or the value of this same key in the XMP metadata, whichever is latest.
IG_PDF_doc_get_layer	Gets the layer with a specified index.
IG_PDF_doc_get_layer_count	Gets the layer count for the document.
IG_PDF_doc_get_page_count	Gets the number of pages in the document.
IG_PDF_doc_set_info	Sets the value of a key in a document's Info dictionary.
IG_PDF_doc_print	Prints PDF pages from a PDF document allowing the user to specify options such as page size, rotation, fit mode.
IG_PDF_doc_create_wordfinder	Creates a word finder that is used to extract text in the host encoding from a PDF file.
IG_PDF_doc_create_wordfinder_ucs	Creates a word finder that is used to extract text in the host encoding from a PDF file.
IG_PDF_doc_get_new_crypt_handler	Gets the specified document's new security handler (that is, the security handler that will be used after the document is saved).
IG_PDF_doc_get_new_security_data	Gets the security data structure for the specified document's new security handler.
IG_PDF_doc_get_new_security_info	Gets the security information from the specified document's new security handler.
IG_PDF_doc_get_security_data	Gets the security data structure for the specified document's new security handler.
IG_PDF_doc_perm_request	Checks the permissions associated with the specified document using the latest permissions format, and determines whether the requested operation is allowed for the specified object in the document.
IG_PDF_doc_set_new_crypt_handler	Sets specified document's new security handler (the security handler that will be used after the document is saved).

IG_PDF_doc_set_new_security_data	Sets the security data structure for the specified document's new security handler.

IG_PDF_doc_create

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create(
HMIGEAR hMPIDoc);
```

Arguments:

hMPIDoc	ImageGear document to attach a PDF document to

Description:

This function creates a PDF document and attaches it to `HMIGEAR`. Doesn't do anything in case if hMPIDoc is already vector document.

To obtain a handle to the PDF document use:

```
   HIG_PDF_DOC hPDFDoc = (HIG_PDF_DOC)NULL;
   IG_mpi_info_get( hMPIDoc, IG_MP_DOCUMENT, &hPDFDoc,
sizeof(hPDFDoc) );
```

Return Value:

Error count.

IG_PDF_doc_create_new_page

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create_new_page(
HMIGEAR hMPIDoc,
LONG nAfterPage,
LPAT_PDF_FIXEDRECT lpMediaBox );
```

Arguments:

hMPIDoc	The document in which the page is created.
nAfterPage	The page number after which the new page is inserted. The first page is 0. Use IG_PDF_BEFORE_FIRST_PAGE to insert the new page at the beginning of a document.
lpMediaBox	Rectangle specifying the page's media box, specified in user space coordinates.

Description:

This function creates a new PDF page for the hMPIDoc. The new PDF page is created at the specified position.

NOTE: The previous value (if any) is not deleted with the IG_image_delete function. The size of the multi-page image is not changed, so that page arrays is not expanded when nAfterPage is greater than pageCount-1.

To obtain a handle to the PDF page use:

```
HIGEAR hNewPage = NULL;
IG_mpi_page_get(m_hMPDoc, nAfterPage+1, &hNewPage);

HIG_PDF_PAGE hNewPDFPage = NULL;
IG_vector_data_get( hNewPage, (LPVOID*)&hNewPDFPage );
```

Return Value:

Error count.

IG_PDF_doc_delete_pages

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_delete_pages(
   HIG_PDF_DOC hDoc,
   LONG nStartPage,
   LONG nPageCount
   );
```

Arguments:

hDoc	The document from which pages are deleted.
nStartPage	The page number of the first page to delete. The first page is 0.
nPageCount	The number of pages to delete.

Description:

Deletes the specified pages.

Return Value:

Error count.

IG_PDF_doc_insert_pages

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_insert_pages(
   HIG_PDF_DOC hDoc,
   LONG nAfterThisPage,
   HIG_PDF_DOC hDoc2,
   LONG nStartPage,
   LONG nPageCount,
   WORD nInsertFlags
   );
```

Arguments:

hDoc	The document into which pages are inserted.
nAfterThisPage	The page number in hDoc after which pages from hDoc2 are inserted. The first page is 0. If IG_PDF_BEFORE_FIRST_PAGE is used, the pages are inserted before the first page in hDoc . Use IG_PDF_LAST_PAGE to insert pages after the last page in hDoc.
hDoc2	The document containing the pages that are inserted into hDoc.
nStartPage	The page number of the first page in hDoc2 to insert into hDoc. The first page is 0.
nPageCount	The number of pages in hDoc2 to insert into hDoc. Use IG_PDF_ALL_PAGES to insert all pages from hDoc2 into hDoc.
nInsertFlags	Flags that determine what additional information is copied from hDoc2 into hDoc. An OR of enumIGPDFInsertFlags constants: ■ IG_PDF_INSERT_BOOKMARKS - Inserts bookmarks as well as pages. The bookmark tree of hDoc2 is merged into the bookmark tree of hDoc by copying it as a new first-level subtree of hDoc's bookmark tree root, of which it becomes the last child. If hDoc has no bookmark tree, it acquires one identical to the bookmark tree from hDoc2. ■ IG_PDF_INSERT_THREADS - Inserts threads as well as pages. ■ IG_PDF_INSERT_ALL - Inserts document data from pages.

Description:

This function inserts nPageCount pages from hDoc2 into hDoc. All annotations, and anything else associated with the page (such as a thumbnail image) are copied from the hDoc2 pages to the new pages in hDoc. This method does not insert pages, if hDoc is equal to hDoc2. The nInsertFlags controls whether bookmarks and threads are inserted along with the specified pages.

Return Value:

Error count.

IG_PDF_doc_get_bookmark

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_bookmark(
    HIG_PDF_DOC hDoc,
    LPHIG_PDF_BOOKMARK lphRootBookmark
    );
```

Arguments:

hDoc	The document whose root bookmark is obtained.
lphRootBookmark	The document's root bookmark.

Description:

Gets the root of the document's bookmark tree. The `lphRootBookmark` value is valid even if document's bookmark tree is empty.

Return Value:

Error count.

IG_PDF_doc_get_info

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_info(
HIG_PDF_DOC hDoc,
LPCSTR szInfoKey,
LPSTR szBuffer,
LONG nSize,
LPLONG lpnBytes);
```

Arguments:

hDoc	The document whose Info dictionary key is obtained.
szInfoKey	The name of the Info dictionary key whose value is obtained.
szBuffer	Result buffer containing the value associated with infoKey. If buffer is NULL, the method will just return the number of bytes required.
nSize	The maximum number of bytes that can be written into buffer.
lpnBytes	If `szBuffer` is NULL, the number of bytes in the specified key's value. If `szBuffer` is not NULL, returns the number of bytes copied into buffer, excluding the terminating NULL. You must pass at least the length + 1 as the buffer size since the routine adds a '\0' terminator to the data, even though the data is not a C string (it can contain embedded '\0's).

Description:

Gets the value of a key in a document's Info dictionary, or the value of this same key in the XMP metadata, whichever is latest.

See Section 10.2.1 in the PDF Reference for information about Info dictionaries. All values in the Info dictionary should be strings; other data types such as numbers and Booleans should not be used as values in the Info dictionary.

Users may define their own Info dictionary entries. In this case, it is strongly recommended that the key have the developer's prefix assigned by the Adobe Solutions Network.

Return Value:

Error count.

IG_PDF_doc_get_layer

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_layer(
   HIG_PDF_DOC hDoc,
   UINT nIndex,
   LPHIG_PDF_LAYER lphLayer
   );
```

Arguments:

hDoc	Document whose layer is obtained.
nIndex	Index of the layer to obtain
lphLayer	obtained layer

Description:

Gets the layer with a specified index.

Return Value:

Error count.

IG_PDF_doc_get_layer_count

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_layer_count(
   HIG_PDF_DOC hDoc,
   LPUINT lpnCount
   );
```

Arguments:

hDoc	Document whose layer count is obtained.
lpnCount	Layer count

Description:

Gets the layer count for the document.

Return Value:

Error count.

IG_PDF_doc_get_page_count

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_page_count(
   HIG_PDF_DOC hDoc,
   LPUINT lpnCount
   );
```

Arguments:

hDoc	The document for which the number of pages is obtained.
lpnCount	The number of pages in the document.

Description:

Gets the number of pages in the document.

Return Value:

Error count.

IG_PDF_doc_set_info

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_set_info(
HIG_PDF_DOC hDoc,
LPCSTR szInfoKey,
LPSTR szBuffer,
LONG nSize);
```

Arguments:

hDoc	The document whose Info dictionary key is set.
szInfoKey	The name of the Info dictionary key whose value is set.
szBuffer	Buffer containing the value to associate with `szInfoKey`.
nSize	The number of bytes in buffer.

Description:

Sets the value of a key in a document's Info dictionary.

See Section 10.2.1 on Info dictionaries in the PDF Reference for information about Info dictionaries. All values in the Info dictionary should be strings; other data types such as numbers and Booleans should not be used as values in the Info dictionary. If an info dictionary key is specified that is not currently in the info dictionary, it is added to the dictionary.

Return Value:

Error count.

IG_PDF_doc_print

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_print(
HIG_PDF_DOC hDoc,
LPAT_PDF_PRINTOPTIONS lpPrintOptions);
```

Arguments:

hDoc	Handle to the document to print
lpPrintOptions	Parameters to control printing

Description:

Prints PDF pages from a PDF document allowing the user to specify options such as page size, rotation, fit mode.

Return Value:

Error count.

IG_PDF_doc_create_wordfinder

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create_wordfinder(HIG_PDF_DOC
hDoc ,LPWORD lpOutEncInfo, LPCHAR *lpOutEncVec, LPCHAR
*lpLigatureTbl, SHORT nAlgVersion, WORD nFlags, LPVOID
lpClientData, LPHIG_PDF_WORDFINDER lphWordFinder )
```

Arguments:

hDoc	The document on which the word finder is used.
lpOutEncInfo	Array of 256 flags, specifying the type of character at each position in the encoding. Each flag is an OR of the Character Type Codes. If `lpOutEncInfo` is NULL, the platform's default encoding info is used. Use `lpOutEncInfo` and `lpOutEncVec` together; for every `lpOutEncInfo` use a corresponding `lpOutEncVec` to specify the character at that position in the encoding.
lpOutEncVec	Array of 256 null-terminated strings that are the glyph names in encoding order. See the discussion of character names in Section 5.3 of the PostScript Language Reference Manual, Third Edition. If `lpOutEncVec` is NULL, the platform's default encoding vector is used. Use this parameter with `lpOutEncInfo`.
lpLigatureTbl	A null-terminated array of null-terminated strings. Each string is the glyph name of a ligature in the font. When a word contains a ligature, the glyph name of the ligature is substituted for the ligature (for example, ff is substituted for the ff ligature). If ligatureTbl is NULL, a default ligature table is used, containing the following ligatures: fi, ff, fl, ffi, ffl, ch, cl, ct, ll, ss, fs, st, oe, OE.
nAlgVersion	The version of the word-finding algorithm to use.
nFlags	Word-finding options that determine the tables filled when using `IG_PDF_wordfinder_acquire_word_list`. Must be an OR of one or more of enumIGPDFWordFlags.
lpClientData	Pointer to user-supplied data to pass to the newly created word finder. Set to NULL.
lphWordFinder	Handle to the new WordFinder

Description:

Creates a word finder that is used to extract text in the host encoding from a PDF file.

NOTE: The word finder also extracts text from Form XObjects that are executed in the page contents. For information about Form XObjects, see Section 4.9 in the PDF Reference.

This method also works for non-Roman (CJK or Chinese-Japanese-Korean) viewers. In this case, words are extracted to the host encoding. Users desiring Unicode output must use `IG_PDF_doc_create_wordfinder_ucs`, which does the extraction for Roman or non-Roman text.

The type of WordFinder determines the encoding of the string returned by `IG_PDF_word_get_string`. For instance, if `IG_PDF_doc_create_wordfinder_ucs` is used to create the word finder, `IG_PDF_word_get_string` returns only Unicode.

For CJK viewers, words are stored internally using CID encoding. For more information on CIDFonts and related topics, see Section 5.6 in the PDF Reference. For detailed information on CIDFonts, see Technical Note #5092, CID-Keyed Font Technology Overview, and Technical Note #5014, Adobe CMap and CIDFont Files Specification.

Return Value:

Error count.

IG_PDF_doc_create_wordfinder_ucs

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_PDF_doc_create_wordfinder_ucs(HIG_PDF_DOC hDoc, SHORT
nAlgVersion, WORD nFlags, LPVOID lpClientData,
LPHIG_PDF_WORDFINDER lphWordFinder )
```

Arguments:

hDoc	The document on which the word finder is used.
nAlgVersion	The version of the word-finding algorithm to use.
nFlags	Word-finding options that determine the tables filled when using `IG_PDF_wordfinder_acquire_word_list`. Must be an OR of one or more of enumIGPDFWordFlags.
lpClientData	Pointer to user-supplied data to pass to the newly created word finder. Set to NULL.
lphWordFinder	Handle to the new WordFinder.

Description:

Creates a word finder that is used to extract text in the host encoding from a PDF file.

NOTE: The word finder also extracts text from Form XObjects that are executed in the page contents. For information about Form XObjects, see Section 4.9 in the PDF Reference.

This method also works for non-Roman (CJK or Chinese-Japanese-Korean) viewers. In this case, words are extracted to the host encoding. Users desiring Unicode output must use `IG_PDF_doc_create_wordfinder_ucs`, which does the extraction for Roman or non-Roman text.

The type of WordFinder determines the encoding of the string returned by `IG_PDF_word_get_string`. For instance, if `IG_PDF_doc_create_wordfinder_ucs` is used to create the word finder, `IG_PDF_word_get_string` returns only Unicode.

For CJK viewers, words are stored internally using CID encoding. For more information on CIDFonts and related topics, see Section 5.6 in the PDF Reference. For detailed information on CIDFonts, see Technical Note #5092, CID-Keyed Font Technology Overview, and Technical Note #5014, Adobe CMap and CIDFont Files Specification.

Return Value:

Error count.

IG_PDF_doc_get_new_crypt_handler

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_new_crypt_handler(
   HIG_PDF_DOC hDoc,
```

```
    LPHIG_PDF_ATOM lphCryptHandler
    );
```

Arguments:

hDoc	The document whose new security handler is obtained.
lphCryptHandler	The PDF atom corresponding to the name of the document's new security handler. Returns IG_PDF_ATOM_NULL if the document does not have a new security handler.

Description:

Gets the specified document's new security handler (that is, the security handler that will be used after the document is saved). If the document does not have a new security handler, returns the document's current security handler.

Return Value:

Error count.

IG_PDF_doc_get_new_security_data

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_new_security_data(
    HIG_PDF_DOC hDoc,
    LPAT_PDF_SECURITYDATA* lppSecData
    );
```

Arguments:

hDoc	The document whose new security data structure is obtained.
lppSecData	The security data structure for the document's new security handler.

Description:

Gets the security data structure for the specified document's new security handler. Use `IG_PDF_doc_get_security_data` to get the security data structure for the document's current security handler.

NOTE: The password strings in PDF are padded or truncated to exactly 32 bytes. If the password string is more than 32 bytes long, used only its first 32 bytes; if it is less than 32 bytes long, it padded by appending the required number of additional

bytes from the beginning of the following padding string: < 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

Return Value:

Error count.

IG_PDF_doc_get_new_security_info

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_new_security_info(
    HIG_PDF_DOC hDoc,
    LPUINT lpnSecInfo
    );
```

Arguments:

hDoc	The document whose new security information is obtained.
lpnSecInfo	The document's new security information. The OR value of the enumIGPDFSecurityInfoFlags.

Description:

Gets the security information from the specified document's new security handler. No permissions are required to call this method.

Return Value:

Error count.

IG_PDF_doc_get_security_data

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_security_data(
    HIG_PDF_DOC hDoc,
    LPAT_PDF_SECURITYDATA* lppSecData
    );
```

Arguments:

hDoc	The document whose security data structure is obtained.
lppSecData	A pointer to the document's current security data structure.

Description:

Gets the security data structure for the specified document's current security handler. Use `IG_PDF_doc_get_new_security_data` to get the data structure for the document's new security handler.

NOTE: The password strings in PDF are padded or truncated to exactly 32 bytes. If the password string is more than 32 bytes long, used only its first 32 bytes; if it is less than 32 bytes long, it padded by appending the required number of additional bytes from the beginning of the following padding string: < 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

Return Value:

Error count.

IG_PDF_doc_perm_request

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_perm_request(
   HIG_PDF_DOC hDoc,
   UINT nReqObj,
   UINT nReqOpr,
   LPVOID lpAuthData,
   LPSHORT lpnReqStatus
   );
```

Arguments:

hDoc	The document whose permissions are being requested.
nReqObj	The target object of the permissions request.
nReqOpr	The target operation of the permissions request.
lpAuthData	A pointer to an authorization data (password string).
lpnReqStatus	One of enumIGPDFPermReqStatus request status constants.

Description:

Checks the permissions associated with the specified document using the latest permissions format, and determines whether the requested operation is allowed for the specified object in the document.

Return Value:

Error count.

IG_PDF_doc_set_new_crypt_handler

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_set_new_crypt_handler(
    HIG_PDF_DOC hDoc,
    HIG_PDF_ATOM hNewCryptHandler
    );
```

Arguments:

hDoc	The document whose new security handler is set.
hNewCryptHandler	The PDF atom for the name of the new security handler to use for the document. Use IG_PDF_ATOM_NULL to remove security from the document.

Description:

Sets specified document's new security handler (the security handler that will be used after the document is saved). This method returns with no action if the new security handler is the same as the old one.

Return Value:

Error count.

IG_PDF_doc_set_new_security_data

Group: Document

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_set_new_security_data(
   HIG_PDF_DOC hDoc,
   LPAT_PDF_SECURITYDATA lpSecData
   );
```

Arguments:

hDoc	The document whose new security data structure is set.
lppSecData	Pointer to the new security data structure to set for doc.

Description:

Sets the security data structure for the specified document's new security handler. The new security handler must have been previously set using `IG_PDF_doc_set_new_crypt_handler`.

Return Value:

Error count.

HIG_PDF_FONT

Type: Font Object

Description:

Handle to a PDF font object that is used to draw text on a page. A HIG_PDF_FONT has a number of attributes, including an array of widths, the character encoding.

Class Members:

IG_PDF_font_get_bbox	Gets a Type 3 font's bounding box
IG_PDF_font_get_charset	Gets the font's character set.
IG_PDF_font_get_cid_systeminfo	Gets Registry and Ordering information for a CIDFont.
IG_PDF_font_get_cid_system_supplement	Gets the SystemSupplement number of a CIDFont.
IG_PDF_font_get_descendant	Gets a Type 0 font's descendant
IG_PDF_font_get_encoding_index	Gets a font's encoding index.
IG_PDF_font_get_encoding_name	Gets a string representing a font's encoding.
IG_PDF_font_get_font_matrix	Gets a font's matrix
IG_PDF_font_get_metrics	Gets a font's metrics
IG_PDF_font_get_name	Gets the name of a font.
IG_PDF_font_get_subtype	Gets a font's subtype.
IG_PDF_font_get_widths	Gets the advance width of every glyph in a font.
IG_PDF_font_is_embedded	Tests whether the specified font is embedded in the PDF file
IG_PDF_font_set_metrics	Sets a font's metrics
IG_PDF_font_translate_string	Translates a string from the hFont's encoding into host encoding.
IG_PDF_font_translate_to_host	Translates a string from the hFont's encoding to host encoding.
IG_PDF_font_translate_to_ucs	Translates a string from whatever encoding the hFont uses to Unicode encoding.
IG_PDF_font_translate_widths	Translates an array of 256 glyph advance widths from their order in the PDF file into host encoding order.

IG_PDF_font_get_bbox

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_bbox(
    HIG_PDF_FONT hFont,
    LPAT_PDF_FIXEDRECT lpBBox
    );
```

Arguments:

hFont	the font whose bounding box is obtained.
lpBBox	the font's bounding box

Description:

Gets a Type 3 font's bounding box, which is the smallest rectangle that would enclose every character in the font if they were overlaid and painted.

IG_PDF_font_get_charset

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_charset(
    HIG_PDF_FONT hFont,
    LPAT_INT lpCharSet
    );
```

Arguments:

hFont	the font whose character set is obtained
lpCharSet	the font's character set

Description:

Gets the font's character set. For non-Roman character set viewers, call IG_PDF_font_get_encoding_name()instead.

IG_PDF_font_get_cid_systeminfo

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_cid_systeminfo(
    HIG_PDF_FONT hFont,
    LPHIG_PDF_ATOM lphCIDSystemInfo
    );
```

Arguments:

hFont	the font whose Registry and Ordering information is obtained
lphCIDSystemInfo	CIDFont's Registry and Ordering information.

Description:

Gets Registry and Ordering information for a CIDFont. IG_PDF_font_get_cid_systeminfo()takes either a Type 0 font or descendant font (CIDType0 or CIDType2) as an argument. This information is always present for any Type 0 font; the actual registry ordering information is a part of the descendant font.

IG_PDF_font_get_cid_system_supplement

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_cid_system_supplement(
   HIG_PDF_FONT hFont,
   LPLONG lpCIDSystemSupplement
   );
```

Arguments:

hFont	the font whose SystemSupplement field is obtained
lpCIDSystemSupplement	the SystemSupplement field from the CIDFont

Description:

Gets the SystemSupplement number of a CIDFont.

IG_PDF_font_get_descendant

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI (
   HIG_PDF_FONT hFont,
   LPHIG_PDF_FONT lphFont
   );
```

Arguments:

hFont	the font whose descendant is obtained
lphFont	the font's descendant font

Description:

Gets a Type 0 font's descendant, which may be a CIDType0 or CIDType2 font.

IG_PDF_font_get_encoding_index

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_encoding_index(
   HIG_PDF_FONT hFont,
   LPLONG lpIndex
   );
```

Arguments:

hFont	the font whose encoding index is obtained
lpIndex	a font encoding index.

Description:

Gets a font's encoding index. For non-Roman character set viewers, it is not appropriate to call this method. Call IG_PDF_font_get_encoding_name() instead.

IG_PDF_font_get_encoding_name

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_encoding_name(
   HIG_PDF_FONT hFont,
   LPBYTE *EncodingName
   );
```

Arguments:

hFont	the font whose encoding name is obtained
EncodingName	string representing the font's encoding

Description:

Gets a string representing a font's encoding. Use IG_PDF_font_get_encoding_index() to get encoding information for Roman viewers.

IG_PDF_font_get_font_matrix

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_font_matrix(
   HIG_PDF_FONT hFont,
   LPAT_PDF_FIXEDMATRIX lpMatrix
   );
```

Arguments:

hFont	the font whose matrix is obtained
lpMatrix	font's matrix

Description:

Gets a font's matrix, which specifies the transformation from character space to text space. This is only valid for Type 3 fonts.

IG_PDF_font_get_metrics

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_metrics(
   HIG_PDF_FONT hFont,
   LPAT_PDF_FONT_METRICS lpFontMetrics
   );
```

Arguments:

hFont	the font whose metrics are obtained
lpFontMetrics	the font's metrics

Description:

Gets a font's metrics, which provide the information needed to create a substitute multiple master font when the original font is unavailable.

IG_PDF_font_get_name

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_name(
   HIG_PDF_FONT hFont,
   LPSTR buffer,
   LONG bufSize,
   LPLONG lpCharacterNum
   );
```

Arguments:

hFont	the font whose name is obtained
buffer	buffer into which the font's name is stored. The client may pass NULL to obtain the buffer size, then call the method with a buffer of the appropriate size.
bufSize	length of buffer, in bytes.
lpCharacterNum	the number of characters in the font name.

Description:

Gets the name of a font. The behavior depends on the font type; for a Type 3 font it gets the value of the Name key in a PDF Font resource. For other types it gets the value of the BaseFont key in a PDF font resource.

IG_PDF_font_get_subtype

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_subtype(
   HIG_PDF_FONT hFont,
```

```
    LPHIG_PDF_ATOM lphSubType
    );
```

Arguments:

hFont	the font whose subtype is obtained
lphSubType	the font's subtype

Description:

Gets a font's subtype.

IG_PDF_font_get_widths

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_widths(
   HIG_PDF_FONT hFont,
   LPSHORT lpWidths
   );
```

Arguments:

hFont	the font whose glyph advance widths are obtained
lphSubType	An array of glyph advance widths, measured in character space units. Unencoded code points will have a width of zero. For non-Roman character set viewers, an array for a single byte range (0 through 255).

Description:

Gets the advance width of every glyph in a font. The advance width is the amount by which the current point advances when the glyph is drawn. The advance width may not correspond to the visible width of the glyph (for example, a glyph representing an accent mark might have an advance width of zero so that characters can be drawn under it). For this reason, the advance width cannot be used to determine the glyphs' bounding boxes. For non-Roman character set viewers, this method gets the width for a single byte range (0 through 255).

IG_PDF_font_is_embedded

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_is_embedded(
   HIG_PDF_FONT hFont,
```

```
    LPAT_PDF_BOOL lpbIsEmbedded
    );
```

Arguments:

hFont	the font to test
lpbIsEmbedded	returns true if the font is embedded in the file, false otherwise

Description:

Tests whether the specified font is embedded in the PDF file (that is, the font is stored as a font file, which is a stream embedded in the PDF file). Only Type 1 and TrueType fonts can be embedded.

IG_PDF_font_set_metrics

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_set_metrics(
    HIG_PDF_FONT hFont,
    LPAT_PDF_FONT_METRICS lpFontMetrics
    );
```

Arguments:

hFont	the font whose metrics are being set
lpFontMetrics	Pointer to a AT_PDF_FONT_METRICS structure containing the font's metrics

Description:

Sets a font's metrics, which provide the information needed to create a substitute multiple master font when the original font is unavailable. This method can only be used on Type 1, multiple master Type 1, and TrueType fonts; it cannot be used on Type 3 fonts.

IG_PDF_font_translate_string

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_string(
    HIG_PDF_FONT hFont,
    LPSTR inP,
```

```
    LPSTR outP,
    LONG len,
    LPAT_PDF_BOOL lpbTableExists
    );
```

Arguments:

hFont	the font (and hence, encoding) that inP uses
inP	the string to translate
outP	the translated string. outP may point to the same buffer as inP, to allow inplace translation
len	the length of inP and outP
lpbTableExists	returns true if an XlateTable exists in the font, false otherwise. If no XlateTable exists in the font, outP is not written

Description:

Translates a string from the hFont's encoding into host encoding. If any characters cannot be represented in host encoding, they are replaced with space characters. If no XlateTable exists in the font, the function returns false and outP is not written. For non-Roman character set viewers, it is not appropriate to call this method.

IG_PDF_font_translate_to_host

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_to_host(
    HIG_PDF_FONT hFont,
    LPSTR inP,
    LONG inLen,
    LPSTR outP,
    LONG outLen,
    LPLONG lpLen
    );
```

Arguments:

hFont	the font used in the input string inP
inP	pointer to the string to translate
inLen	length of the inP buffer, in bytes
outP	pointer to the translated string
outLen	the length of the outP buffer, in bytes
lpLen	number of bytes in the translated string outP

Description:

Translates a string from the hFont's encoding to host encoding.

IG_PDF_font_translate_to_ucs

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_to_ucs(
   HIG_PDF_FONT hFont,
   LPSTR inP,
   LONG inLen,
   LPSTR outP,
   LONG outLen,
   LPLONG lpLen
   );
```

Arguments:

hFont	the font of the input string inP
inP	pointer to the string to translate
inLen	length of the inP buffer, in bytes
outP	pointer to the translated string
outLen	the length of the outP buffer, in bytes
lpLen	number of bytes in the translated string outP

Description:

Translates a string from whatever encoding the hFont uses to Unicode encoding.

IG_PDF_font_translate_widths

Group: Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_widths(
   HIG_PDF_FONT hFont,
   LPSHORT inP,
   LPSHORT outP
   );
```

Arguments:

hFont	the font whose glyph widths are translated
inP	array of glyph advance widths to rearrange
outP	rearranged array of glyph advance widths

Description:

Translates an array of 256 glyph advance widths from their order in the PDF file into host encoding order. If the widths are already in host encoding order, the widths are merely copied. All un-encoded code points are given a width of zero. For non-Roman character set viewers, it is not appropriate to call this method.

HIG_PDF_LAYER

Type: Page Object

Description:

Handle to the PDF layer object which represents a named object whose state can be toggled in a UI to affect changes in visibility of content.

Class Members:

IG_PDF_layer_create	Creates new layer (optional-content group) object in the document.
IG_PDF_layer_get_name	Gets the layer name
IG_PDF_layer_set_name	Sets the new layer name
IG_PDF_layer_get_current_state	Gets the current ON-OFF state of the layer object
IG_PDF_layer_set_current_state	Sets the current ON-OFF state of the layer object
IG_PDF_layer_get_initial_state	Gets the initial ON-OFF state of the layer object
IG_PDF_layer_set_initial_state	Sets the initial ON-OFF state of the layer object
IG_PDF_layer_has_usage_info	Tests whether a layer object is associated with a Usage dictionary
IG_PDF_layer_get_usage_info	Gets usage information from a layer object
IG_PDF_layer_set_usage_info	Sets a Usage dictionary entry in a layer object.
IG_PDF_layer_get_intent	Gets the intent list for a layer.
IG_PDF_layer_set_intent	Sets the Intent entry in a layer's dictionary.
IG_PDF_layer_get_unique_id	Sets a unique ID.
IG_PDF_layer_delete	Deletes a PDF layer object.
IG_PDF_layer_release	Releases a layer object.
IG_PDF_layer_remove	Destroys layer (optional-content group) object.

IG_PDF_layer_create

Group: Layer

Declaration:

```
IG_PDF_layer_create(
   HIG_PDF_DOC hDoc,
   LPSTR lpTextBuf,
   AT_DWORD nTextLen,
   LPHIG_PDF_LAYER lphLayer)
```

Arguments:

hDoc	The document in which the group is used.
lpTextBuf	The name of the layer.
nTextLen	Length of the lpTextBuf.
lphLayer	The newly created group object.

Description:

Creates new layer (optional-content group) object in the document.

Return Value:

The newly created group object.

IG_PDF_layer_get_name

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_name(
   HIG_PDF_LAYER hLayer,
   LPSTR lpTextBuf,
   UINT nBufSize,
   LPUINT lpnTextLen
   );
```

Arguments:

hLayer	layer whose name is obtained
lpTextBuf	pointer to a buffer for text string
nBufSize	maximum size of the buffer
lpnTextLen	Actual length of the name string copied into the buffer

Description:

This function gets the layer name.

Return Value:

Error count.

IG_PDF_layer_set_name

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_name(
   HIG_PDF_LAYER hLayer,
   LPSTR lpTextBuf,
   UINT nTextLen
   );
```

Arguments:

hLayer	layer whose name is set
lpTextBuf	pointer to a string buffer
nBufSize	length of the string buffer

Description:

This function sets the new layer name.

Return Value:

Error count.

IG_PDF_layer_get_current_state

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_current_state(
   HIG_PDF_LAYER hLayer,
   LPAT_PDF_BOOL lpbState
   );
```

Arguments:

hLayer	Layer whose current state is obtained.
lpbState	Returns TRUE if the state is ON, FALSE if it is OFF.

Description:

This function gets the current ON-OFF state of the layer object.

Return Value:

Error count.

IG_PDF_layer_set_current_state

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_current_state(
   HIG_PDF_LAYER hLayer,
   AT_PDF_BOOL bState
   );
```

Arguments:

hLayer	Layer whose current state is set.
bState	New state.

Description:

This function sets the current ON-OFF state of the layer object.

Return Value:

Error count.

IG_PDF_layer_get_initial_state

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_initial_state(
   HIG_PDF_LAYER hLayer,
   LPAT_PDF_BOOL lpbState
   );
```

Arguments:

hLayer	Layer whose initial state is obtained.
lpbState	Returns TRUE if the state is ON, FALSE if it is OFF.

Description:

This function gets the initial ON-OFF state of the layer object.

Return Value:

Error count.

IG_PDF_layer_set_initial_state

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_initial_state(
   HIG_PDF_LAYER hLayer,
   AT_PDF_BOOL bState
   );
```

Arguments:

hLayer	Layer whose initial state is set.
bState	The new state.

Description:

This function sets the initial ON-OFF state of the layer object.

Return Value:

Error count.

IG_PDF_layer_has_usage_info

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_has_usage_info(
   HIG_PDF_LAYER hLayer,
   LPAT_PDF_BOOL lpbHasUsage
   );
```

Arguments:

hLayer	Layer object.
lpbHasUsage	Returns TRUE if the layer has a Usage dictionary, FALSE otherwise.

Description:

This function verifies whether a layer object is associated with a Usage dictionary.

Return Value:

Error count.

IG_PDF_layer_get_usage_info

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_usage_info(
   HIG_PDF_LAYER hLayer,
   HIG_PDF_ATOM hUsageKey,
   LPHIG_PDF_BASOBJ lphUsageInfo
   );
```

Arguments:

hLayer	Layer object whose usage information is obtained.
hUsageKey	The usage key in the usage dictionary entry. Possible key values are: ■ CreatorInfo ■ Language ■ Export ■ Zoom ■ Print ■ View ■ User ■ PageElement
lphUsageInfo	The usage information associated with the given key in the Usage dictionary for the layer, or a NULL if the operation fails (because the layer is malformed or has no dictionary, or because the dictionary has no entry corresponding to the given key).

Description:

This function gets usage information from a layer object. A Usage dictionary entry provides more specific intended usage information than an intent entry.

Return Value:

Error count.

IG_PDF_layer_set_usage_info

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_usage_info(
   HIG_PDF_LAYER hLayer,
   HIG_PDF_ATOM hUsageKey,
   HIG_PDF_BASOBJ hUsageInfo
   );
```

Arguments:

hLayer	Layer object whose usage information is set.
hUsageKey	The usage key in the usage dictionary entry. Possible key values are: ■ CreatorInfo ■ Language ■ Export ■ Zoom ■ Print ■ View ■ User ■ PageElement
hUsageInfo	The usage information to associate with the key.

Description:

This function sets a Usage dictionary entry in a layer object. The entry associates usage information with an entry key for retrieval. If a dictionary does not exist, the method creates one.

A Usage dictionary entry provides more specific intended usage information than an intent entry.

The usage value can act as a kind of metadata, describing the sort of things that belong to the layer: for example, text in French, fine detail on a map, or a watermark. The usage values can also be used by the AutoState mechanism to make decisions about what layers should be on and what layers should be off.

Return Value:

Error count.

IG_PDF_layer_get_intent

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_intent(
   HIG_PDF_LAYER hLayer,
   LPHIG_PDF_ATOM lphIntent,
   UINT nSize,
   LPUINT lpnLen
   );
```

Arguments:

hLayer	Layer object whose intent is obtained.
lphIntent	Returns array of intent entries.
nSize	Maximal size of lphIntent.
lpnLen	Actual number of intent entries copied to lphIntent.

Description:

This function gets the intent list for a layer. Intent is an atom value broadly describing the intended use, either View or Design. A layer's content is considered to be optional (that is, the layer's state is considered in its visibility) if any intent in its list matches an intent of the context. The intent list of the context is usually set from the intent list of the document configuration.

Return Value:

Error count.

IG_PDF_layer_set_intent

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_intent(
    HIG_PDF_LAYER hLayer,
    LPHIG_PDF_ATOM lphIntent,
    UINT nLen
    );
```

Arguments:

hLayer	Layer object whose intent is set.
lphIntent	New Intent entry value, an array of intent entries (atoms).
nLen	Number of intent entries in lphIntent.

Description:

This function sets the Intent entry in a layer's dictionary. Intent is an atom value broadly describing the intended use, either View or Design. A layer's content is considered to be optional (that is, the layer's state is considered in its visibility) if any intent in its list matches an intent of the context. The intent list of the context is usually set from the intent list of the document configuration.

Return Value:

Error count.

IG_PDF_layer_get_unique_id

Group: Layer

Declaration:

```
IG_PDF_layer_get_unique_id(
   HIG_PDF_LAYER hLayer,
   LPUINT lpnUniqueId);
```

Arguments:

hLayer	Layer object.
lpnUniqueId	The unique identifier.

Description:

Returns some 32bit integer that is unique for all Layer objects. In other words, it is guaranteed that there cannot be two Layer objects with the same UniqueIds.

Can be used for Layer objects' identification.

Return Value:

An unique identifier of this Layer Object.

IG_PDF_layer_delete

Group: Layer

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_delete(
   HIG_PDF_LAYER hLayer
   );
```

Arguments:

hLayer	Layer object to delete.

Description:

(Use the new **IG_PDF_layer_release** function instead)

This function deletes a PDF layer object.

Return Value:

Error count.

IG_PDF_layer_release

Group: Layer

Declaration:

```
IG_PDF_layer_release(
   HIG_PDF_LAYER hLayer)
```

Arguments:

hLayer	Layer object to release.

Description:

(The same as the old method **IG_PDF_layer_delete**).

Releases the native object and frees memory.

Return Value:

Nothing.

IG_PDF_layer_remove

Group: Layer

Declaration:

```
IG_PDF_layer_remove(
   HIG_PDF_LAYER hLayer)
```

Arguments:

hLayer	The layer object.

Description:

Destroys layer (optional-content group) object. This does not delete any content.

HIG_PDF_PAGE

Type: Page Object

Description:

Handle to the PDF page object. A single page in the PDF representation of a document. A page contains a series of objects representing the objects drawn on the page (Graphic), a list of resources used in drawing the page, annotations (Annotation), an optional thumbnail image of the page, and the beads used in any articles that occur on the page.

Class Members:

IG_PDF_page_get_content	Creates HIG_PDE_CONTENT from HIG_PDF_PAGE.
IG_PDF_page_get_crop_box	Gets the crop box for the page.
IG_PDF_page_get_rotation	Gets the rotation for the page.
IG_PDF_page_make_color_separations	Separates hPage's DeviceN colorants into individual layers.
IG_PDF_page_set_content	Sets the page's PDF content back into the HIG_PDF_PAGE object, using the same compression filters with which the content was previously encoded.
IG_PDF_page_release_content	Decrements HIG_PDF_PAGE's PDF content internal reference count.
IG_PDF_page_get_annotation_count	Gets the number of annotations on a page.

IG_PDF_page_get_content

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_get_content(
HIG_PDF_PAGE hPage,
LPHIG_PDE_CONTENT lpnContent);
```

Arguments:

hPage	The page whose content object is acquired
lpnContent	PDE content return value

Description:

Creates `HIG_PDE_CONTENT` from `HIG_PDF_PAGE`. The PDF content is cached, so that subsequent calls on the same page return the same PDF content.

Return Value:

Error count.

IG_PDF_page_get_crop_box

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI i_IG_PDF_page_get_crop_box(
   HIG_PDF_PAGE hPage,
   LPAT_PDF_FIXEDRECT lpFixedBox
);
```

Arguments:

hPage	The page whose crop box is obtained.
lpFixedBox	The crop box

Description:

Gets the crop box for the page.

IG_PDF_page_get_rotation

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI i_IG_PDF_page_get_rotation(
   HIG_PDF_PAGE hPage,
   LPSHORT lpRotation
);
```

Arguments:

hPage	The page whose rotation is obtained.
lpRotation	The rotation.

Description:

Gets the rotation for the page

IG_PDF_page_make_color_separations

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_make_color_separations(
HIG_PDF_PAGE hPage,
LPHIG_PDF_ATOM pColorChannels,
UINT nColorChannelsNum,
HIG_PDF_DOC hDoc,
LONG nAfterPage
);
```

Arguments:

hPage	The page whose content object is acquired.
pColorChannels	Atom array containing the colorant names to separate. Example: "C", "M", "Y", "K", "PANTONE 300 C", etc.
nColorChannelsNum	Number of elemens in pColorChannels.
hDoc	Output PDF document.
nAfterPage	The page number in the output PDF document after which the result page is inserted.

Description:

Separates `hPage`'s DeviceN colorants into individual layers. The result page is inserted in `hDoc` after the page with index `nAfterPage`.

Return Value:

Error count.

Example:

```
HIG_PDF_ATOM* pColorChannels = new HIG_PDF_ATOM[2];
IG_PDF_atom_from_string("Y", &pColorChannels[0]);
IG_PDF_atom_from_string("PANTONE 300 C", &pColorChannels[1]);
```

```
nErrCount +=
IG_PDF_page_make_color_separations(GetCurPDFPage(),
pColorChannels, 2, GetPDFDoc(), nPageCount-2);
delete pColorChannels;
```

IG_PDF_page_set_content

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_set_content(
HIG_PDF_PAGE hPage);
```

Arguments:

hPage	The page whose content object is set

Description:

Sets the page's PDF content back into the HIG_PDF_PAGE object, using the same compression filters with which the content was previously encoded.

Return Value:

Error count.

IG_PDF_page_release_content

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_release_content(
HIG_PDF_PAGE hPage);
```

Arguments:

hPage	The page whose content object's use count is decremented

Description:

Decrements HIG_PDF_PAGE's PDF content internal reference count. The content is not automatically deleted when the reference count becomes zero — it remains in the cache until the cache slot is needed for another HIG_PDF_PAGE. Thus, you do not need to keep a content acquired for performance reasons.

Return Value:

Error count.

IG_PDF_page_get_annotation_count

Group: Page

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_get_annotation_count(
HIG_PDF_PAGE hPage,
LPUINT lpnCount);
```

Arguments:

hPage	The page for which the number of annotations is obtained
lpnCount	The number of annotations return value

Description:

Gets the number of annotations on a page. Annotations associated with pop-up windows (such as strikeouts) are counted as two annotations. Widget annotations (form fields) are included in the count.

Return Value:

Error count.

HIG_PDF_WORDFINDER

Type: Word Finder Object

Description:

Handle to the PDF word finder object. Extracts words from a PDF file, and enumerates the words on a single page or on all pages in a document.

Class Members:

IG_PDF_wordfinder_acquire_wordlist	Finds all words on the specified page.
IG_PDF_wordfinder_release_wordlist	Releases the word list for a given page.
IG_PDF_wordfinder_get_word	Gets the word in the word list obtained using IG_PDF_wordfinder_acquire_wordlist.
IG_PDF_wordfinder_delete	Deletes a word finder.

IG_PDF_wordfinder_acquire_wordlist

Group: Word Finder

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_acquire_wordlist(
HIG_PDF_WORDFINDER hWordFinder, LONG nPageNumber, LPLONG
lpnWordCount );
```

Arguments:

hWordFinder	The word finder used to acquire the word list.
nPageNumber	The page number for which words are found. First page is 0.
lpnWordCount	The number of words found on the page.

Description:

Finds all words on the specified page.

Only words within or partially within the page's crop box are enumerated. Words outside the crop box are skipped.

There can be only one word list in existence at a time; clients must release the previous word list, using IG_PDF_wordfinder_release_wordlist, before creating a new one.

Return Value:

Error count.

IG_PDF_wordfinder_release_wordlist

Group: Word Finder

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_release_wordlist(
HIG_PDF_WORDFINDER hWordFinder, LONG nPageNumber );
```

Arguments:

hWordFinder	The word finder object.
nPageNumber	The page number for which the word list is released.

Description:

Releases the word list for a given page. Use this to release a list created by `IG_PDF_wordfinder_acquire_wordlist` when you are done using this list.

Return Value:

Error count.

IG_PDF_wordfinder_get_word

Group: Word Finder

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_get_word (
HIG_PDF_WORDFINDER hWordFinder, WORD nFlags, LONG nIndex,
LPHIG_PDF_WORD lphWord );
```

Arguments:

hWordFinder	The word finder object.
nFlags	Word-finding options. Must be an OR of one or more of enumIGPDFWordFlags.
nIndex	The index of the word to obtain. The first word on a page has an index of zero. Words are counted in PDF order.
lphWord	A handle to the word object. Returns NULL when the end of the list is reached.

Description:

Gets the word in the word list obtained using `IG_PDF_wordfinder_acquire_wordlist`.

Return Value:

Error count.

IG_PDF_wordfinder_delete

Group: Word Finder

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_delete(
HIG_PDF_WORDFINDER hWordFinder);
```

Arguments:

hWordFinder	Word finder object to delete.

Description:

Deletes a word finder. Use this when you are done extracting text in a file.

Return Value:

Error count.

HIG_PDF_WORD

Type: Word Object

Description:

Handle to the PDF word object. A word in a PDF file. Each word contains a sequence of characters in one or more styles (see Style).

Class Members:

IG_PDF_word_get_char_offset	Returns a word's character offset from the beginning of its page.
IG_PDF_word_get_char_style	Returns a PDF Style object for the specified style in a word.
IG_PDF_word_get_length	Gets the number of bytes in a word.
IG_PDF_word_get_quad	Gets the specified word's quad, specified in user space coordinates.
IG_PDF_word_get_quad_count	Gets the number of quads in a word.
IG_PDF_word_get_string	Gets a word's text and also converts ligatures to their constituent characters.
IG_PDF_word_get_style_transition	Gets the locations of style transitions in a word.
IG_PDF_word_delete	Deletes a word object.

IG_PDF_word_get_char_offset

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_PDF_word_get_char_offset(HIG_PDF_WORD hWord, LPWORD
lpnCharOffset);
```

Arguments:

hWord	The word whose character offset is obtained.
lpnCharOffset	The word's character offset. On multi-byte systems, it points to the first byte.

Description:

Returns a word's character offset from the beginning of its page.

Return Value:

Error count.

IG_PDF_word_get_char_style

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_char_style(
HIG_PDF_WORDFINDER hWordFinder,
HIG_PDF_WORD hWord, LONG nIndex, LPHIG_PDF_STYLE lphStyle );
```

Arguments:

hWordFinder	A word finder object.
hWord	The word whose character style is obtained.
nIndex	The index of the style to obtain. The first style in a word has an index of zero.
lphStyle	The obtained style in the word. Returns NULL if nIndex is greater than the number of styles in the word.

Description:

Returns a PDF Style object for the specified style in a word.

Return Value:

Error count.

IG_PDF_word_get_length

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_length(HIG_PDF_WORD hWord,
LPWORD lpnLength);
```

Arguments:

hWord	The word whose character count is obtained.
lpnLength	The number of characters in the word.

Description:

Gets the number of bytes in a word. This method also works on non-Roman systems.

Return Value:

Error count.

IG_PDF_word_get_quad

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_quad( HIG_PDF_WORD hWord,
SHORT nIndex, LPAT_PDF_FIXEDQUAD lpQuad, LPAT_PDF_BOOL
lpbHasQuad );
```

Arguments:

hWord	The word whose quad is obtained.
nIndex	The index of the quad to obtain. The first quad in a word has an index of zero.
lpQuad	Pointer to the word's quad, specified in user-space coordinates.
lpbHasQuad	True if the word has the specified quad, false otherwise.

Description:

Gets the specified word's quad, specified in user space coordinates. See `IG_PDF_word_get_quad_count` for a description of a quad.

The quad's height is the height of the font's bounding box, not the height of the tallest character used in the word. The font's bounding box is determined by the glyphs in the font that extend farthest above and below the baseline; it often extends somewhat above the top of "A" and below the bottom of "y."

The quad's width is determined from the characters actually present in the word.

As an example, the quads for the words "AWAY" and "away" have the same height, but generally do not have the same width unless the font is a mono-spaced font (a font in which all characters have the same width).

Return Value:

Error count.

IG_PDF_word_get_quad_count

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_quad_count (HIG_PDF_WORD
hWord, LPWORD lpnQuadCount);
```

Arguments:

hWord	The word whose quad count is obtained.
lpnQuadCount	The number of quads in the word.

Description:

Gets the number of quads in a word. A quad is a quadrilateral bounding a contiguous piece of a word. Every word has at least one quad. A word has more than one quad, for example, if it is hyphenated and split across multiple lines or if the word is set on a curve rather than on a straight line.

Return Value:

Error count.

IG_PDF_word_get_string

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_string (
HIG_PDF_WORD hWord, LPCHAR lpString, LONG nLen );
```

Arguments:

hWord	The word whose string is obtained.
lpString	The word string. The encoding of the string is the encoding used by the PDF WordFinder that supplied the PDF Word. For instance, if `IG_PDF_doc_create_wordfinder_ucs` is used to create the word finder, this function returns only Unicode.
nLen	Length of string, in bytes. Up to nLen characters of word will be copied into `lpString`. If `lpString` is long enough, it will be null-terminated.

Description:

Gets a word's text and also converts ligatures to their constituent characters. The string to return includes any word break characters (such as space characters) that follow the word, but not any that precede the word. The characters that are treated as word breaks are defined in the `outEncInfo` parameter of `IG_PDF_doc_create_wordfinder` method.

This method produces a string in whatever encoding the PDF Word uses, for both Roman and non-Roman systems.

Return Value:

Error count.

IG_PDF_word_get_style_transition

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_style_transition(
HIG_PDF_WORD hWord, LPSHORT lpTransTbl, SHORT nSize, LPSHORT
lpnStTrCount );
```

Arguments:

hWord	The word whose style transition list is obtained.
lpTransTbl	(Filled by the method) Array of style transitions. Each element is the character offset in word where the style changes. The offset specifies the first character in the word that has the new style. The first character in a word has an offset of zero.
nSize	Number of entries that `lpTransTbl` can hold. The word is searched only until this number of style transitions has been found.
lpnStTrCount	Number of style transition offsets copied to `lpTransTbl`.

Description:

Gets the locations of style transitions in a word. Every word has at least one style transition, at character position zero in the word.

Return Value:

Error count.

IG_PDF_word_delete

Group: Word

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_delete(
HIG_PDF_WORD hWord);
```

Arguments:

hWord	Word object to delete.

Description:

Deletes a word object.

Return Value:

Error count.

HIG_PDF_STYLE

Type: Style Object

Description:

Handle to the PDF style object; provides access to information about the fonts, font sizes, and colors used in a Word.

Class Members:

IG_PDF_style_get_color	Gets a style's color.
IG_PDF_style_get_font	Gets the specified style's font.
IG_PDF_style_get_font_size	Get a style's font size.
IG_PDF_style_delete	Deletes a PDF style object.

IG_PDF_style_get_color

Group: Style

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_get_color(
   HIG_PDF_STYLE hStyle,
   LPAT_PDF_COLORVALUE lpColor
   );
```

Arguments:

hStyle	The style whose color is obtained
lpColor	The style's color

Description:

Gets a style's color.

IG_PDF_style_get_font

Group: Style

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_get_font(
   HIG_PDF_STYLE hStyle,
```

```
    LPHIG_PDF_FONT lphFont
    );
```

Arguments:

hStyle	The style whose font is obtained
lphFont	The font for the specified style

Description:

Gets the specified style's font.

IG_PDF_style_get_font_size

Group: Style

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_PDF_style_get_font_size(HIG_PDF_STYLE hStyle,
LPAT_PDF_FIXED lpnFontSize);
```

Arguments:

hStyle	The style whose font size is obtained.
lpnFontSize	A style's font size.

Description:

Get a style's font size.

Return Value:

Error count.

IG_PDF_style_delete

Group: Style

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_delete(
HIG_PDF_STYLE hStyle );
```

Arguments:

hStyle	Style object to delete.

Description:

Deletes a PDF style object.

Return Value:

Error count.

HIG_PDF_SYSFONT

Type: System Font Object

Description:

Handle to the system font object. A reference to a font installed in the host system. SysFont methods allow you to list the fonts available in the host system and to find a font in the system that matches a PDE Font, if it is present.

Class Members:

IG_PDF_sysfont_enumerate	Enumerates all of the system fonts with a user-supplied procedure.
IG_PDF_sysfont_find	Finds a system font that matches the requested attributes.
IG_PDF_sysfont_find_for_pdefont	Gets platform-specific data for use by user interface code.
IG_PDF_sysfont_get_platform_data	Gets platform-specific data for use by user interface code.
IG_PDF_sysfont_get_attrs	Gets the attributes of a system font.
IG_PDF_sysfont_get_cid_system_info	Derives the registry, ordering, and supplement information of a multi-byte system font.
IG_PDF_sysfont_get_create_flags	This function obtains `lpnFlags` that can be passed to `IG_PDE_font_create_from_sysfont_and_encoding`.
IG_PDF_sysfont_get_info	Gets high-level information about a system font.
IG_PDF_sysfont_get_name	Gets the PostScript or TrueType styled name for a system font.
IG_PDF_sysfont_get_widths	Gets the widths of a single byte encoded system font.
IG_PDF_sysfont_release_platform_data	Releases platform-specific data for the specified `hSysFont`.
IG_PDF_sysfont_release	Releases a system font object.

IG_PDF_sysfont_enumerate

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_enumerate(
```

```
LPFNIG_PDF_SYSFONT_ENUMPROC lpfnEnumProc,
LPVOID lpClientData );
```

Arguments:

lpfnEnumProc	User-supplied callback to call once for each system font. Enumeration continues until all fonts have been enumerated, or until `lpfnEnumProc` returns false.
lpClientData	Pointer to user-supplied data to pass to `lpfnEnumProc` each time it is called.

Description:

Enumerates all of the system fonts with a user-supplied procedure.

The SysFont parameters must be copied during the enumeration if they are needed beyond the `lpfnEnumProc`.

Developers should not assume that the `lpfnEnumProc` will get called. If no system fonts are found, `lpfnEnumProc` is never called.

Return Value:

Error count.

IG_PDF_sysfont_find

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_find(
LPAT_PDE_FONTATTRS lpAttrs,
LONG nFlags,
LPHIG_PDF_SYSFONT lphSysFont);
```

Arguments:

lpAttrs	Pointer to `AT_PDE_FONTATTRS` structure with the attributes of the font you are searching for.
nFlags	Bit field comprised of `enumIGPDFSysFontMatchFlags` values. Passing zero matches font by name only.
lphSysFont	The desired system font.

Description:

Finds a system font that matches the requested attributes.

Return Value:

Error count.

IG_PDF_sysfont_find_for_pdefont

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_find_for_pdefont(
HIG_PDE_FONT hFont,
LONG nFlags,
LPHIG_PDF_SYSFONT lphSysFont );
```

Arguments:

hFont	A PDE Font whose matching system font is found.
nFlags	Bit field comprised of `enumIGPDFSysFontMatchFlags` values. Passing zero matches font by name only.
lphSysFont	The desired system font.

Description:

Finds a system font that matches the requested `hFont`.

Return Value:

Error count.

IG_PDF_sysfont_get_platform_data

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_platform_data(
HIG_PDF_SYSFONT hSysFont,
LPAT_PDF_SYSFONT_PLATDATA *lpPlatData );
```

Arguments:

hSysFont	A SysFont object referencing a system font.
lpPlatData	Pointer to a LPAT_PDF_SYSFONT_PLATDATA containing information relating to a system font. Returns NULL if out of memory.

Description:

Gets platform-specific data for use by user interface code. Must be released when finished by `IG_PDF_sysfont_release_platform_data`.

Return Value:

Error count.

IG_PDF_sysfont_get_attrs

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_attrs(
HIG_PDF_SYSFONT hSysFont,
LPAT_PDE_FONTATTRS lpAttrs );
```

Arguments:

hSysFont	A SysFont object referencing a system font.
lpAttrs	Pointer to AT_PDE_FONTATTRS with the attributes of a system font.

Description:

Gets the attributes of a system font.

The attributes will be returned in the buffer pointed to by `lpAttrs`.

This call can be expensive to execute, as it may involve parsing the font in order to determine attributes.

Return Value:

Error count.

IG_PDF_sysfont_get_cid_system_info

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_cid_system_info(
HIG_PDF_SYSFONT hSysFont,
LPHIG_PDF_ATOM lphRegistry,
LPHIG_PDF_ATOM lphOrdering,
LPLONG lpnSupplement );
```

Arguments:

hSysFont	A SysFont object referencing a multi-byte system font.
lphRegistry	The PDF atom representing the CID Font's registry information, as in "Adobe".
lphOrdering	The PDF atom representing the CID Font's ordering information, for example, "Japan1".
lpnSupplement	The SystemSupplement field from the CID Font.

Description:

Derives the registry, ordering, and supplement information of a multi-byte system font. This information can be used to create a PDE Font from a system font.

Return Value:

Error count.

IG_PDF_sysfont_get_create_flags

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_create_flags(
HIG_PDF_SYSFONT hSysFont,
HIG_PDF_SYSENCODING hSysEncoding,
LPLONG lpnFlags );
```

Arguments:

hSysFont	A SysFont object.
hSysEncoding	A SysEncoding object.
lpnFlags	Create flags that can be passed to `IG_PDE_font_create_from_sysfont_and_encoding`. If the combination of `hSysFont` and `hSysEncoding` is not allowed, it is set to `-1`.

Description:

This function obtains `lpnFlags` that can be passed to `IG_PDE_font_create_from_sysfont_and_encoding`. If the combination of `hSysFont` and `hSysEncoding` is not allowed, it is set to `-1`.

Return Value:

Error count.

IG_PDF_sysfont_get_info

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_info(
HIG_PDF_SYSFONT hSysFont,
LPAT_PDE_FONT_INFO lpInfo );
```

Arguments:

hSysFont	A SysFont object referencing a system font whose information is obtained.
lpInfo	Pointer to `AT_PDE_FONT_INFO` structure to fill with font information for `hSysFont`.

Description:

Gets high-level information about a system font.

Return Value:

Error count.

IG_PDF_sysfont_get_name

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_name(
HIG_PDF_SYSFONT hSysFont,
LPHIG_PDF_ATOM lphName );
```

Arguments:

hSysFont	A SysFont object referencing a system font whose name is obtained.
lphName	The PDF atom for the system font's name.

Description:

Gets the PostScript or TrueType styled name for a system font.

Return Value:

Error count.

IG_PDF_sysfont_get_widths

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_widths(
HIG_PDF_SYSFONT hSysFont,
LPSHORT lpWidths,
LPAT_PDF_FIXED mmDesignVector );
```

Arguments:

hSysFont	A SysFont object referencing a system font whose widths are obtained.
lpWidths	Pointer to widths array. `lpWidths` must have room for 256 entries.
mmDesignVector	If `hSysFont` is a multiple master font, points to the design vector, whose length must equal the number of design axes of `hSysFont`.

Description:

Gets the widths of a single byte encoded system font.

Return Value:

Error count.

IG_PDF_sysfont_release_platform_data

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_release_platform_data(
HIG_PDF_SYSFONT hSysFont,
LPAT_PDF_SYSFONT_PLATDATA lpPlatData );
```

Arguments:

hSysFont	A SysFont object referencing a system font.
lpPlatData	Pointer to AT_PDF_SYSFONT_PLATDATA containing platform-specific data to release.

Description:

Releases platform-specific data for the specified `hSysFont`.

Return Value:

Error count.

IG_PDF_sysfont_release

Group: System Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_release(
HIG_PDF_SYSFONT hSysFont );
```

Arguments:

hSysFont	A system font to release.

Description:

Releases a system font object.

Return Value:

Error count.

HIG_PDF_SYSENCODING

Type: System Encoding Object

Description:

Handle to the system encoding object. Provides system encoding for a PDF file.

Class Members:

IG_PDF_sysencoding_create_from_base_name	Create an encoding object from base name.
IG_PDF_sysencoding_create_from_cmap_name	Create an encoding object from a PDF CMap name.
IG_PDF_sysencoding_create_from_code_page	Create an encoding object from a PDF CMap name.
IG_PDF_sysencoding_get_writing_mode	Returns writing mode in `lpnWritingMode`.
IG_PDF_sysencoding_is_identity	Returns in `lpbResult` true for Identity-H or Identity-V encoding; false otherwise.
IG_PDF_sysencoding_is_multibyte	Returns in `lpbResult` true for CMap encoding; false otherwise.
IG_PDF_sysencoding_release	Release an encoding object.

IG_PDF_sysencoding_create_from_base_name

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_create_from_base_name(
HIG_PDF_ATOM hBaseEncName,
LPCSTR *lpDiffEnc,
LPHIG_PDF_SYSENCODING lphSysEncoding );
```

Arguments:

hBaseEncName	The base encoding. See Section 5.5.5 in the PDF Reference.
lpDiffEnc	Array of 256 const char* describing the differences from the encoding specified by `hBaseEncName`. May be NULL.
lphSysEncoding	New encoding object.

Description:

Create an encoding object from base name.

Return Value:

Error count.

IG_PDF_sysencoding_create_from_cmap_name

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_create_from_cmap_name(
HIG_PDF_ATOM hCMapName,
LPHIG_PDF_SYSENCODING lphSysEncoding );
```

Arguments:

hCMapName	The CMap name.
lphSysEncoding	New encoding object.

Description:

Create an encoding object from a PDF CMap name.

Return Value:

Error count.

IG_PDF_sysencoding_create_from_code_page

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_create_from_code_page(
LONG nCodePage,
SHORT nWritingMode,
```

```
LPHIG_PDF_SYSENCODING lphSysEncoding );
```

Arguments:

nCodePage	The code page character-mapping construct. One of `enumIGPDFCodePages` values.
nWritingMode	0 for horizontal writing, 1 for vertical writing.
lphSysEncoding	New encoding object.

Description:

Create an encoding object from a code page.

Return Value:

Error count.

IG_PDF_sysencoding_get_writing_mode

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_get_writing_mode(
HIG_PDF_SYSENCODING hSysEncoding,
LPSHORT lpnWritingMode);
```

Arguments:

hSysEncoding	An encoding object.
lpnWritingMode	0 for horizontal writing and 1 for vertical writing.

Description:

Returns writing mode in `lpnWritingMode`. 0 for horizontal writing and 1 for vertical writing.

Return Value:

Error count.

IG_PDF_sysencoding_is_identity

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_is_identity(
HIG_PDF_SYSENCODING hSysEncoding,
LPAT_BOOL lpbResult );
```

Arguments:

hSysEncoding	An encoding object.
lpbResult	True for Identity-H or Identity-V encoding; false otherwise.

Description:

Returns in `lpbResult` true for Identity-H or Identity-V encoding; false otherwise.

Return Value:

Error count.

IG_PDF_sysencoding_is_multibyte

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_is_multibyte(
HIG_PDF_SYSENCODING hSysEncoding,
LPAT_BOOL lpbResult );
```

Arguments:

hSysEncoding	An encoding object.
lpbResult	True for CMap encoding; false otherwise

Description:

Returns in `lpbResult` true for CMap encoding; false otherwise.

Return Value:

Error count.

IG_PDF_sysencoding_release

Group: System Encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_release(
HIG_PDF_SYSENCODING hSysEncoding )
```

Arguments:

hSysEncoding	An encoding object to release.

Description:

Release an encoding object.

Return Value:

Error count.

Page Editing Objects and Elements

Objects described in this chapter provide an easy access to PDF page contents. With this group of objects you can treat a page's contents as a list of objects - rather than having to manipulate the content stream's PDF marking operators. The editing (or PDE) objects provide an easy access to PDF page contents. PDE objects are meant to be used in conjunction with the General and Base Object methods for manipulating PDF documents.

The PDE objects are splitted in two groups: PDE elements and PDE objects. A page display list is represented as a PDE object that contains PDE elements. The page's content is usually treated as a list of PDE elements. Each PDE element is a path, text, image, form, or a marked content place or container of PDE Elements. Using the PDE Objects and methods you can add or remove objects inside a PDE Objects. You can also change attributes of Elements in a PDE Object, such as a bounding box, a text font, or a clipping path.

The PDE objects are used to create the PDE elements and provide specific information for the elements. For example, PDE Image, which is an element, usually has or is created based on a PDE ColorSpace, which is an object. PDE text, which is an element, usually has or is created based on a PDE Font, which is an object.

This API is meant to be used in conjunction with basic and general objects.

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

Page Editing Objects and Elements	
HIG_PDE_OBJECT	Object – base interface for all the PDE objects.

HIG_PDE_CONTENT	Content – modifiable content of a PDF page, which contains elements. Content may be obtained from an existing page or from a Form XObject.
HIG_PDE_COLORSPACE	ColorSpace – a reference to a color space used on a page in a PDF file. The color space is part of the graphics state attributes of a PDE Element.
HIG_PDE_FONT	Font – A reference to a font used on a page in a PDF file. It may be equated with a font in the system.
HIG_PDE_ELEMENT	Element - base interface for the elements of a page display list (PDE content) and for clip objects. The general PDE element methods allow you to get and set general element properties.
HIG_PDE_CLIP	Clip – a list of elements containing a list of Paths and Texts that describe a clip state. Clips can be created and built up with PDE Clip methods. Any PDE Element object can have Clip associated with it. Clip objects can contain PDE Containers and PDE Groups to an arbitrary level of nesting. This allows PDE Containers to be used to mark clip objects. PDE Groups inside PDE Clips that contain at least one PDE Text and no PDE Paths have a special meaning. All PDE Text objects contained in such a PDE Group are considered to be part of the same BT/ET block. This means that the union of these PDE Texts makes up a single clipping path—as opposed to the intersection of the PDE Texts. See Section 5.3 in the PDF reference for more information about BT/ET block.
HIG_PDE_CONTAINER	Container - a group of elements on a page in a PDF file. In the PDF file, containers are delimited by Marked Content BMC/EMC or BDC/EMC pairs. Every container has a Marked Content tag associated with it. In addition to grouping a set of elements, a BDC/EMC pair specifies a property list to be associated with the grouping. Thus a container corresponding to a BDC/EMC pair also has a property list dictionary associated with it. See Section 10.5 in the PDF reference for more information about Marked Content operators.

HIG_PDE_FORM	Form - an element that corresponds to an instance of XObject Form on a page (or other containing stream such as another XObject Form or annotation form). The context associated with this instance includes the actual stream that represents the XObject Form and the initial conditions of the graphics state. The latter consists of the transformation matrix, initial color values, and so forth. It is possible to have two Forms that refer to the same XObject Form. The forms will exist at different places on the same page, depending on the transformation matrix. They may also have different colors or line stroking parameters. In the case of a transparency group, the opacity is specified in the gstate. Within a Form, each element has its own gstate (or is a container, place, or group object). These gstates are independent of the parent Form gstate. Form elements may have their own opacity. Content may be obtained from a Form to edit the form's display list.
HIG_PDE_GROUP	Group - an in-memory representation of objects in Content. It has no state and is not represented in any way in a content stream (that is, Content). When used in a Clip, this object is used to associate Text objects into a single clipping object.
HIG_PDE_IMAGE	Image - an element that contains an Image XObject or in-line image. You can associate data or a stream with an image.
HIG_PDE_PATH	Path – an element that contains a path. Path objects can be stroked, filled, and/or serve as a clipping path.
HIG_PDE_PLACE	Place – an element that marks a place on a page in a PDF file. In a PDF file, a place is represented by the MP or DP Marked Content operators. Marked content is useful for adding structure information to a PDF file. For instance, a drawing program may want to mark a point with information, such as the start of a path of a certain type. Marked content provides a way to retain this information in the PDF file. A DP operator functions the same as the MP operator and, in addition, allows a property list dictionary to be associated with a place.
HIG_PDE_POSTSCRIPT	PostScript – an element representing in-line or XObject pass-through PostScript object. XObject PostScripts are listed in page XObject resources.
HIG_PDE_SOFTMASK	SoftMask– an object for creating and manipulating a soft mask in a PDF file.
HIG_PDE_SHADING	Shading – an element that represents smooth shading.

HIG_PDE_TEXT	Text – an element representing text. It is a container for text as show strings or as individual characters. Each sub-element may have different graphics state properties. However, the same clip applies to all sub-elements of a Text. Also, the char path of a Text can be used to represent a clip.
HIG_PDE_TEXTITEM	TextItem – a PDE element representing a text object.
HIG_PDE_XGROUP	XGroup – a transparency (XGroup) resource.
HIG_PDE_XOBJECT	XObject – an element representing an arbitrary XObject.

HIG_PDE_OBJECT

Type: Abstract PDE Object

Description:

Handle to abstract PDE object. Base interface for all the PDE objects.

Class Members:

IG_PDE_object_get_type	Gets the type of an object.

IG_PDE_object_get_type

Group: Abstract PDE

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_object_get_type(
HIG_PDE_OBJECT hObject,
LPLONG lpnType);
```

Arguments:

hObject	The object whose type is obtained.
lpnType	Type return value

Description:

Gets the type of an object.

Return Value:

Error count.

HIG_PDE_CONTENT

Type: PDE Content Object

Description:

Handle to the PDE content object. Modifiable content of a PDF page, which contains elements. Content may be obtained from an existing page or from a Form XObject.

Class Members:

IG_PDE_content_create	Creates empty PDEContent.
IG_PDE_content_get_element	Obtains requested element from content.
IG_PDE_content_add_element	Inserts an element into content.
IG_PDE_content_remove_element	Removes an element from content.
IG_PDE_content_get_element_count	Gets the number of elements in a content.
IG_PDE_content_get_default_color_space	Gets a default color space from `hContent`.
IG_PDE_content_get_attrs	Gets the attributes of a content.

IG_PDE_content_create

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_create(
   LPHIG_PDE_CONTENT lphContent
   );
```

Arguments:

lphContent	The created PDEContent.

Description:

This method creates empty PDEContent.

IG_PDE_content_get_element

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_element(
HIG_PDE_CONTENT hContent,
LONG nIndex,
LPHIG_PDE_ELEMENT lphElement);
```

Arguments:

hContent	Content to obtain from
nIndex	Index of element to obtain
lphElement	PDE element return value

Description:

Obtains requested element from content.

Return Value:

Error count.

IG_PDE_content_add_element

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_add_element(
HIG_PDE_CONTENT hContent,
LONG nAfterIndex,
HIG_PDE_ELEMENT hElement);
```

Arguments:

hContent	Content to which `hElement` is added
nAfterIndex	Location after which `hElement` is added. Should be `IG_PDE_BEFORE_FIRST` to add to the beginning of the display list.
hElement	The element to add to the content.

Description:

Inserts an element into content.

Return Value:

Error count.

IG_PDE_content_remove_element

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_remove_element(
HIG_PDE_CONTENT hContent,
LONG nIndex);
```

Arguments:

hContent	Content to remove from
nIndex	Index of element to remove

Description:

Removes an element from content.

Return Value:

Error count.

IG_PDE_content_get_element_count

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_element_count(
HIG_PDE_CONTENT hContent,
LPUINT lpnCount);
```

Arguments:

hContent	Content
lpnCount	The number of elements return value

Description:

Gets the number of elements in a content.

Return Value:

Error count.

IG_PDE_content_get_default_color_space

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_default_color_space(
HIG_PDE_CONTENT hContent,
HIG_PDF_ATOM hColorSpaceName,
LPHIG_PDE_COLORSPACE lphDefaultColorSpace);
```

Arguments:

hContent	Content to whose default color space is obtained
hColorSpaceName	An atom for the name of the desired color space. Must be an atom for one of DefaultRGB, DefaultCMYK, or DefaultGray.
lphDefaultColorSpace	The desired color space in `hContent`. Returns NULL if `hColorSpaceName` does not correspond to a known default, such as DefaultRGB.

Description:

Gets a default color space from `hContent`.

See Section 4.5.4 in the PDF Reference for more information about default color spaces.

Return Value:

Error count.

IG_PDE_content_get_attrs

Group: PDE Content

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_attrs (
HIG_PDE_CONTENT hContent,
LPAT_PDE_CONTENTATTRS lpAttrs );
```

Arguments:

hContent	Content whose attributes are obtained
lpAttrs	Pointer to a AT_PDE_CONTENTATTRS structure to fill with the attributes of the content.

Description:

Gets the attributes of a content.

Return Value:

Error count.

HIG_PDE_COLORSPACE

Type: PDE Color Space Object

Description:

Handle to the PDE color space object. A reference to a color space used on a page in a PDF file. The color space is part of the graphics state attributes of a PDE Element.

Class Members:

IG_PDE_colorspace_create	Creates a new color space object of the specified type.
IG_PDE_colorspace_get_base_name	Gets the name of the base color space.
IG_PDE_colorspace_get_base_color_components	Gets the number of components in the base color space of an indexed color space.
IG_PDE_colorspace_get_ctable	Gets the component information for an indexed color space.
IG_PDE_colorspace_get_hival	Gets the highest index for the color lookup table for an indexed color space.
IG_PDE_colorspace_get_name	Gets the name of a color space object.
IG_PDE_colorspace_get_color_components	Calculates the number of components in a color space.
IG_PDE_colorspace_release	Releases a color space object.

IG_PDE_colorspace_create

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_create(
HIG_PDF_ATOM hFamily,
LPAT_PDE_COLORDATA lpColorData,
LPHIG_PDE_COLORSPACE lphColorSpace );
```

Arguments:

hFamily	Supports the following PDF color spaces: Device-dependent names: DeviceCMYK, DeviceGray, DeviceN, or DeviceRGB. Device-independent names: CalGray, CalRGB, Lab, or ICCBased. Special names: Indexed, Pattern, or Separation.
lpColorData	Color data for the type of color space you want to create.
lphColorSpace	Handle to the new color space

Description:

Creates a new color space object of the specified type.

Return Value:

Error count.

IG_PDE_colorspace_get_base_name

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_base_name (
HIG_PDE_COLORSPACE hColorSpace,
LPHIG_PDF_ATOM lphBaseName );
```

Arguments:

hColorSpace	The base color space.
lphBaseName	The atom for the name of the base color space.

Description:

Gets the name of the base color space. This is a helper routine for indexed color spaces.

Return Value:

Error count.

IG_PDE_colorspace_get_base_color_components

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_base_color_components
(
HIG_PDE_COLORSPACE hColorSpace,
LPLONG lpnColorComponents);
```

Arguments:

hColorSpace	The base color space.
lpnColorComponents	Number of components in `hColorSpace`.

Description:

Gets the number of components in the base color space of an indexed color space.

For example, for [/Indexed /DeviceRGB...], the number of components is 3.

Return Value:

Error count.

IG_PDE_colorspace_get_ctable

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_ctable (
HIG_PDE_COLORSPACE hColorSpace,
LPBYTE lpColorTable);
```

Arguments:

hColorSpace	The color space whose component information table is obtained.
lpColorTable	The color lookup table, which is nColorComponents * (nHiVal + 1) bytes long, where nColorComponents = number of components in the base `hColorSpace`. Each entry in the table contains `nColorComponents` bytes, and the table is indexed 0 to nHiVal, where nHiVal is the highest index in the color table. The table is indexed from 0 to nHiVal, thus the table contains nHiVal + 1 entries.

Description:

Gets the component information for an indexed color space.

Return Value:

Error count.

IG_PDE_colorspace_get_hival

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_hival (
HIG_PDE_COLORSPACE hColorSpace,
LPLONG lpnHiVal);
```

Arguments:

hColorSpace	An indexed color space.
lpnHiVal	The highest index (nHiVal) in the color lookup table.

Description:

Gets the highest index for the color lookup table for an indexed color space. Since the color table is indexed from zero to nHiVal, the actual number of entries is nHiVal + 1.

Return Value:

Error count.

IG_PDE_colorspace_get_name

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_name (
HIG_PDE_COLORSPACE hColorSpace,
LPHIG_PDF_ATOM lphName );
```

Arguments:

hColorSpace	A color space object.
lphName	The color space object's name. Supports the following PDF color spaces: Device-dependent names: DeviceCMYK, DeviceGray, DeviceN, or DeviceRGB. Device-independent names: CalGray, CalRGB, Lab, or ICCBased. Special names: Indexed, Pattern, or Separation.

Description:

Gets the name of a color space object.

Return Value:

Error count.

IG_PDE_colorspace_get_color_components

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_color_components (
HIG_PDE_COLORSPACE hColorSpace,
LPLONG lnpColorComponents);
```

Arguments:

hColorSpace	A color space object.
lpnColorComponents	Number of components in `hColorSpace`. For: DeviceGray, CalGray, Separation: Returns 1. DeviceRGB, CalRGB: Returns 3. DeviceCMYK, Lab: Returns 4. DeviceN, ICCBased: Returns the number of components dependent on the specific color space object. Indexed: Returns 1. Use `IG_PDE_colorspace_get_base_color_components` to get the number of components in the base color space.

Description:

Calculates the number of components in a color space.

Return Value:

Error count.

IG_PDE_colorspace_release

Group: PDE Color Space

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_release(
HIG_PDE_COLORSPACE hColorSpace );
```

Arguments:

hColorSpace	Color space object to release.

Description:

Releases a color space object.

Return Value:

Error count.

HIG_PDE_FONT

Type: PDE Font Object

Description:

Handle to the PDE font object. A reference to a font used on a page in a PDF file. It may be equated with a font in the system.

Class Members:

IG_PDE_font_create	Creates a new PDE Font from the specified parameters.
IG_PDE_font_create_from_sysfont	Creates a PDE Font corresponding to a font in the system.
IG_PDE_font_create_from_sysfont_and_encoding	Create a PDE Font from `hSysFont` and `hSysEncoding`.
IG_PDE_font_create_from_sysfont_with_params	Used to obtain a PDE Font corresponding to a font in the system.
IG_PDE_font_get_attrs	Gets the attributes for a font object.
IG_PDE_font_create_tounicode_now	This function creates the /ToUnicode table.
IG_PDE_font_create_widths_now	This function creates width entries for font.
IG_PDE_font_embed_now	This function embeds font stream.
IG_PDE_font_embed_now_dont_subset	Embeds the given `hFont` inside `hDoc` without creating a subset.
IG_PDE_font_get_create_need_flags	This function returns flags indicating what needs to be done to make `hFont` complete.
IG_PDE_font_get_codebyte_count	Gets the number of bytes comprising the next code in a string of single or multi-byte character codes.
IG_PDE_font_get_onebyte_encoding	Gets an array of delta encodings for the given one byte PDE Font.
IG_PDE_font_get_sysencoding	Gets the system encoding object associated with a font object.
IG_PDE_font_get_sysfont	Gets the system font object associated with a font object.
IG_PDE_font_get_widths	Gets the widths for a font object.
IG_PDE_font_get_widths_now	Gets a Type0 font's width information for only the characters used in the file.
IG_PDE_font_is_embedded	Tests whether a font is an embedded font in the document in which it was created.
IG_PDE_font_is_multibyte	Tests whether a font contains any multi-byte characters.
IG_PDE_font_set_sysencoding	Sets the system encoding object associated with a font object.

IG_PDE_font_set_sysfont	Sets the system font object to be used with a font object that does not currently have a system font associated with it.
IG_PDE_font_subset_now	Subsets a given PDE Font in `hDoc`.
IG_PDE_font_sum_widths	Gets the sum to the widths of `nTextLen characters` from a string of single or multi-byte characters.
IG_PDE_font_translate_glyphids_to_unicode	Translates a string to Unicode values.
IG_PDE_font_release	Releases PDE font object.

IG_PDE_font_create

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create(
LPAT_PDE_FONTATTRS lpAttrs,
LONG nFirstCharIndex,
LONG nLastCharIndex,
LPSHORT lpWidths,
LPSTR *lpEncoding,
HIG_PDF_ATOM hEncodingBaseName,
HIG_PDF_STREAM hFontStm,
LONG nLen1,
LONG nLen2,
LONG nLen3,
LPHIG_PDE_FONT lphFont);
```

Arguments:

lpAttrs	Pointer to AT_PDE_FONTATTRS for the font attributes.
nFirstCharIndex	First character index for the widths array, lpWidths.
nLastCharIndex	Last character index for the widths array, lpWidths.
lpWidths	Widths array.
lpEncoding	Array of 256 glyph names specifying the custom encoding. If any pointer is NULL, no encoding information is written for that entry.
hEncodingBaseName	Encoding base name if the encoding is a custom encoding. If encoding is NULL, encodingBaseName is used as the value of the encoding, and must be one of WinAnsiEncoding, MacRomanEncoding, or MacExpertEncoding. If no encoding value is desired, use IG_PDF_ATOM_NULL.
hFontStm	Stream with font information.
nLen1	Length in bytes of the ASCII portion of the Type 1 font file after it has been decoded. For other font formats, such as TrueType or CFF, only Len1 is used, and it is the size of the font.
nLen2	Length in bytes of the encrypted portion of the Type 1 font file after it has been decoded.
nLen3	Length in bytes of the portion of the Type 1 font file that contains the 512 zeros, plus the cleartomark operator, plus any following data.
lphFont	The specified PDE font.

Description:

Creates a new PDE Font from the specified parameters.

The PDE Font may be represented as an embedded font (a FontFile entry in the font descriptor of the PDF file). To create a PDE Font that is stored as an embedded font, the FontFile stream may be passed in hFontStm, and the nLen1, nLen2, and nLen3 parameters contain the Length1, Length2, and Length3 values of the FontFile stream attributes dictionary. See Section 5.8 in the PDF Reference for more information about embedded fonts.

Call IG_PDE_font_release to dispose of the returned font object when finished with it.

Return Value:

Error count.

IG_PDE_font_create_from_sysfont

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_from_sysfont(
HIG_PDF_SYSFONT hSysFont,
LONG nCreateFlags,
HIG_PDF_ATOM hSnapshotName,
LPAT_PDF_FIXED mmDesignVec,
LPHIG_PDE_FONT lphFont );
```

Arguments:

hSysFont	PDF system font object referencing a system font.
nCreateFlags	Indicates whether to embed the font and whether to subset the font. Must be one of enumIGPDEFontCreateFlags. If you want to subset a font, set both the `IG_PDE_FONT_CREATE_EMBEDDED` and `IG_PDE_FONT_WILL_SUBSET` flags.
hSnapshotName	Name to be associated with this particular instantiation of the PDE Font.
mmDesignVec	Multiple master font design vector.
lphFont	The PDE Font corresponding to `hSysFont`.

Description:

Creates a PDE Font corresponding to a font in the system.

If the font is a multiple master font, `mmDesignVec` points to the design vector, whose length must equal the number of design axes of the font.

The enumIGPDEFontCreateFlags flags `IG_PDE_FONT_CREATE_EMBEDDED` and `IG_PDE_FONT_WILL_SUBSET` must both be set in order to subset a font.

If you create a PDE Font that is subsetted, call `IG_PDE_font_subset_now` on this font afterwards.

Call `IG_PDE_font_release` to dispose of the returned font object when finished with it.

Return Value:

Error count.

IG_PDE_font_create_from_sysfont_and_encoding

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_PDE_font_create_from_sysfont_and_encoding(
HIG_PDF_SYSFONT hSysFont,
HIG_PDF_SYSENCODING hSysEncoding,
HIG_PDF_ATOM hUseThisBaseFont,
LONG nCreateFlags,
LPHIG_PDE_FONT lphFont );
```

Arguments:

hSysFont	PDF system font object referencing a system font.
hSysEncoding	A PDF SysEncoding object.
hUseThisBaseFont	The base font. An error will be set if the base font name passed is a subset name (XXXXXX+FontName) or an empty string.
nCreateFlags	One of the enumIGPDEFontCreateFlags values.
lphFont	The new PDE Font object.

Description:

Create a PDE Font from `hSysFont` and `hSysEncoding`. If it fails, it returns an error. User can call `IG_PDF_sysfont_get_create_flags` to see if the combination of `hSysFont` and `hSysEncoding` makes sense.

Call `IG_PDE_font_release` to dispose of the returned font object when finished with it.

Return Value:

Error count.

IG_PDE_font_create_from_sysfont_with_params

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI
IG_PDE_font_create_from_sysfont_with_params(
HIG_PDF_SYSFONT hSysFont,
LPAT_PDE_FONT_CREATEFROMSYSFONTPARAMS lpParams,
LPHIG_PDE_FONT lphFont );
```

Arguments:

hSysFont	PDF system font object referencing a system font.
lpParams	Pointer to the parameters structure.
lphFont	The new PDE Font object.

Description:

Used to obtain a PDE Font corresponding to a font in the system.

Call `IG_PDE_font_release` to dispose of the returned font object when finished with it.

Return Value:

Error count.

IG_PDE_font_get_attrs

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_attrs(
HIG_PDE_FONT hFont,
LPAT_PDE_FONTATTRS lpAttrs);
```

Arguments:

hFont	PDE font whose attributes are found.
lpAttrs	Font attributes return value

Description:

Gets the attributes for a font object.

Return Value:

Error count.

IG_PDE_font_create_tounicode_now

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_tounicode_now(
HIG_PDE_FONT hFont,
```

```
    HIG_PDF_DOC hDoc );
```

Arguments:

hFont	PDE font to create /ToUnicode table for.
hDoc	The container document

Description:

This function creates the /ToUnicode table. The user can check the return value of `IG_PDE_font_get_create_need_flags` to see if calling of `IG_PDE_font_create_tounicode_now` is needed.

Return Value:

Error count.

IG_PDE_font_create_widths_now

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_widths_now(
   HIG_PDE_FONT hFont,
   HIG_PDF_DOC hDoc );
```

Arguments:

hFont	PDE font to create width entries for.
hDoc	The container document

Description:

This function creates width entries for font. The user can check the return value of `IG_PDE_font_get_create_need_flags` to see if calling of `IG_PDE_font_create_widths_now` is needed.

Return Value:

Error count.

IG_PDE_font_embed_now

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_embed_now(
HIG_PDE_FONT hFont,
HIG_PDF_DOC hDoc );
```

Arguments:

hFont	PDE font to embed
hDoc	The container document

Description:

This function embeds font stream. The user can check the return value of `IG_PDE_font_get_create_need_flags` to see if calling of `IG_PDE_font_embed_now` is needed.

Return Value:

Error count.

IG_PDE_font_embed_now_dont_subset

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_embed_now_dont_subset(
   HIG_PDE_FONT hFont,
   HIG_PDF_DOC hDoc );
```

Arguments:

hFont	PDE font to embed
hDoc	The container document

Description:

Embeds the given `hFont` inside `hDoc` without creating a subset. Use this method instead of `IG_PDE_font_embed_now` if you created font with the `IG_PDE_FONT_WILL_SUBSET` flag but changed your mind.

Return Value:

Error count.

IG_PDE_font_get_create_need_flags

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_create_need_flags(
HIG_PDE_FONT hFont,
LPLONG lpnFlags );
```

Arguments:

hFont	PDE font object
lpnFlags	A value corresponding to `enumIGPDEFontCreateNeedFlags`.

Description:

This function returns flags indicating what needs to be done to make `hFont` complete.

`IG_PDE_FONT_CREATE_NEED_WIDTHS` can be cleared by `IG_PDE_font_create_widths_now`.

`IG_PDE_FONT_CREATE_NEED_TO_UNICODE` can be cleared by `IG_PDE_font_create_tounicode_now`.

`IG_PDE_FONT_CREATE_NEED_EMBED` can be cleared by `IG_PDE_font_create_embed_now`.

Return Value:

Error count.

IG_PDE_font_get_codebyte_count

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_codebyte_count(
HIG_PDE_FONT hFont,
LPBYTE lpText,
LONG nTextLen,
LPUINT lpnCount );
```

Arguments:

hFont	PDE font object
lpText	Pointer into a string of characters.
nTextLen	The length, in bytes, of the string of characters, starting with the character pointed to by `lpText`.
lpnCount	Number of bytes in the next character code pointed to by `lpText`.

Description:

Gets the number of bytes comprising the next code in a string of single or multi-byte character codes.

Return Value:

Error count.

IG_PDE_font_get_onebyte_encoding

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_onebyte_encoding(
HIG_PDE_FONT hFont,
LPHIG_PDF_ATOM lphEncodingDelta,
LPAT_BOOL lpbGotEncodingDelta );
```

Arguments:

hFont	PDE font object
lphEncodingDelta	Pointer to an atom array that is filled with the delta encodings for font. Each entry is the atom for a glyph name that differs from the base encoding. See Section 5.5.5 in the PDF Reference for more information about font encodings. The array must be allocated to hold 256 entries.
lpbGotEncodingDelta	True if encodingDelta is filled, False otherwise.

Description:

Gets an array of delta encodings for the given one byte PDE Font.

Return Value:

Error count.

IG_PDE_font_get_sysencoding

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_sysencoding(
HIG_PDE_FONT hFont,
LPHIG_PDF_SYSENCODING lphSysEncoding );
```

Arguments:

hFont	PDE font whose system encoding is found
lphSysEncoding	The system ensoding object.

Description:

Gets the system encoding object associated with a font object.

Return Value:

Error count.

IG_PDE_font_get_sysfont

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_sysfont(
HIG_PDE_FONT hFont,
LPHIG_PDF_SYSFONT lphSysFont );
```

Arguments:

hFont	PDE font whose system font is found
lphSysFont	The system font object.

Description:

Gets the system font object associated with a font object.

Return Value:

Error count.

IG_PDE_font_get_widths

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_widths(
HIG_PDE_FONT hFont,
LPSHORT lpWidths );
```

Arguments:

hFont	PDE font whose widths are found
lpWidths	Pointer to widths array. `lpWidths` must have room for 256 values. The widths are returned in character space (1000 EM units). An EM is a typographic unit of measurement equal to the size of a font. To convert to text space, divide the value returned by 1000. To convert to user space, multiply the text space value by the font size.

Description:

Gets the widths for a font object.

Return Value:

Error count.

IG_PDE_font_get_widths_now

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_widths_now(
HIG_PDE_FONT hFont,
HIG_PDF_DOC hDoc );
```

Arguments:

hFont	PDE font whose widths are found
hDoc	The container document

Description:

Gets a Type0 font's width information for only the characters used in the file. Call this routine when the font was created with the `IG_PDE_FONT_DEFER_WIDTHS` flag but without the `IG_PDE_FONT_CREATE_EMBEDDED` flag (if the font is to be embedded, call `IG_PDE_font_subset_now`, which also gets the width info).

Return Value:

Error count.

IG_PDE_font_is_embedded

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_is_embedded(
HIG_PDE_FONT hFont,
LPAT_BOOL lpbResult );
```

Arguments:

hFont	PDE font object
lpbResult	True if the font is embedded, False if it is not, or if it was created in one document and embedded in a different document.

Description:

Tests whether a font is an embedded font in the document in which it was created.

Return Value:

Error count.

IG_PDE_font_is_multibyte

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_is_multibyte(
HIG_PDE_FONT hFont,
LPAT_BOOL lpbResult );
```

Arguments:

hFont	PDE font object
lpbResult	True if the font contains any multi-byte characters, False otherwise.

Description:

Tests whether a font contains any multi-byte characters.

Return Value:

Error count.

IG_PDE_font_set_sysencoding

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_set_sysencoding(
HIG_PDE_FONT hFont,
HIG_PDF_SYSENCODING hSysEncoding );
```

Arguments:

hFont	PDE Font whose system encoding is set.
hSysEncoding	The new system encoding object.

Description:

Sets the system encoding object associated with a font object.

NOTE: Changing the system encoding may produce unexpected results.

Return Value:

Error count.

IG_PDE_font_set_sysfont

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_set_sysfont(
HIG_PDE_FONT hFont,
HIG_PDF_SYSFONT hSysFont );
```

Arguments:

hFont	PDE Font whose system font is set.
hSysFont	The new system font object.

Description:

Sets the system font object to be used with a font object that does not currently have a system font associated with it.

Return Value:

Error count.

IG_PDE_font_subset_now

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_subset_now(
HIG_PDE_FONT hFont,
HIG_PDF_DOC hDoc );
```

Arguments:

hFont	PDE font to subset
hDoc	The document whose font is subsetted.

Description:

Subsets a given PDE Font in `hDoc`.

If you created font with `IG_PDE_font_create_from_sysfont`, you must have set both the `IG_PDE_FONT_CREATE_EMBEDDED` and `IG_PDE_FONT_WILL_SUBSET` set in the flags parameter to be able to subset font.

Return Value:

Error count.

IG_PDE_font_sum_widths

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_sum_widths(
HIG_PDE_FONT hFont,
```

```
LPBYTE lpText,
LONG nTextLen,
LPLONG lpnSum );
```

Arguments:

hFont	PDE font object\
lpText	Pointer into a string of characters.
nTextLen	Number of characters in the string.
lpnSum	Width of text string in EM space. (In EM space, the width of “M” is about 1000 EM units).

Description:

Gets the sum to the widths of `nTextLen characters` from a string of single or multi-byte characters.

Return Value:

Error count.

IG_PDE_font_translate_glyphids_to_unicode

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_translate_glyphids_to_unicode(
HIG_PDE_FONT hFont,
LPBYTE lpText,
LONG nTextLen,
LPBYTE lpUniText,
LONG nUniTextLen,
LPLONG lpnResult );
```

Arguments:

hFont	PDE font object
lpText	The string to convert.
nTextLen	The length of `lpText`, in bytes
lpUniText	Buffer to hold the translated string.
nUniTextLen	The size of the `lpUniText` buffer.
lpnResult	0 if the string was successfully translated. If `lpUniText` is too small for the translated string, it returns the number of bytes required.

Description:

Translates a string to Unicode values. The PDE Font must have a **/ToUnicode** table.

Return Value:

Error count.

IG_PDE_font_release

Group: PDE Font

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_release(
HIG_PDE_FONT hFont );
```

Arguments:

hFont	PDE font object to release

Description:

Releases PDE font object.

Return Value:

Error count.

HIG_PDE_ELEMENT

Type: PDE Element Object

Description:

Handle to the PDE element object. Base interface for the elements of a page display list (PDE content) and for clip objects. The general PDE element methods allow you to get and set general element properties.

Class Members:

IG_PDE_element_get_type	Gets the type of an element.
IG_PDE_element_clone	Makes a copy of an element.
IG_PDE_element_is_at_point	Tests whether a point is on an element.
IG_PDE_element_is_at_rect	Tests whether any part of a rectangle is on an element.
IG_PDE_element_get_bbox	Gets the bounding box for an element.
IG_PDE_element_get_clip	Gets the current clip for an element.
IG_PDE_element_get_gstate	Gets the graphics state information for an element.
IG_PDE_element_set_gstate	Sets the graphics state information for an element.
IG_PDE_element_get_matrix	Gets the transformation matrix for an element.
IG_PDE_element_set_matrix	Sets the transformation matrix for an element.
IG_PDE_element_get_dictionary	Returns a Dictionary (OCMD object).
IG_PDE_element_get_unique_id	Returns some 32bit integer that is unique for all Element objects.
IG_PDE_element_has_gstate	Tests if `hElement` has a graphics state information.
IG_PDE_element_release	Release the specified element.

IG_PDE_element_get_type

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_type(
HIG_PDE_ELEMENT hElement,
LPLONG lpnType);
```

Arguments:

hElement	The element whose type is obtained.
lpnType	Type return value

Description:

Gets the type of an element.

Return Value:

Error count.

IG_PDE_element_clone

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_clone(
HIG_PDE_ELEMENT hElement,
LONG nFlags,
LPHIG_PDE_ELEMENT lphCloneElement);
```

Arguments:

hElement	The element to copy.
nFlags	Bit field of `enumIGPDEElementCopyFlags`.
lphCloneElement	A copy of `hElement`

Description:

Makes a copy of an element. The caller is responsible for releasing the copy with `IG_PDE_element_release`.

Return Value:

Error count.

IG_PDE_element_is_at_point

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_is_at_point(
HIG_PDE_ELEMENT hElement,
```

```
LPAT_PDF_FIXEDPOINT lpPoint,
LPAT_BOOL lpbResult);
```

Arguments:

hElement	The element to test. If hElement is a Text or Image, it uses the bounding box of the element to make the check. If the `hElement` is a Path and it is stroked, it checks if the point is on the path. If the `hElement` is a Path and it is filled, it checks if the point is in the fill area, taking into consideration whether it is filled using the non-zero winding number rule or the even-odd rule.
lpPoint	The point, specified in user space coordinates.
lpbResult	TRUE if the point is on the element, FALSE otherwise.

Description:

Tests whether a point is on an element.

Return Value:

Error count.

IG_PDE_element_is_at_rect

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_is_at_rect(
HIG_PDE_ELEMENT hElement,
LPAT_PDF_FIXEDRect lpRect,
LPAT_BOOL lpbResult);
```

Arguments:

hElement	The element to test. If `hElement` is a PDE Text or PDE Image, it uses the bounding box of the PDE Element to make the check. If `hElement` is a PDE Path and it is stroked, it checks if the rectangle is on the path. If `hElement` is a PDE Path and it is filled, it checks if the rectangle is in the fill area, taking into consideration whether it is filled using the non-zero winding number rule or the even-odd rule.
lpRect	The rectangle, specified in user space coordinates.
lpbResult	True if any part of the rectangle is on the element, False otherwise.

Description:

Tests whether any part of a rectangle is on an element.

Return Value:

Error count.

IG_PDE_element_get_bbox

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_bbox(
HIG_PDE_ELEMENT hElement,
LPAT_PDF_FIXEDRECT lpBBox );
```

Arguments:

hElement	An element whose bounding box is obtained.
lpBBox	Pointer to a `AT_PDF_FIXEDRECT` structure specifying the bounding box of `hElement`, specified in user space coordinates.

Description:

Gets the bounding box for an element.

The returned bounding box is guaranteed to encompass the element, but is not guaranteed to be the smallest box that could contain the element. For example, for an arc, `lpBBox` encloses the Bezier control points—not just the curve itself.

Return Value:

Error count.

IG_PDE_element_get_clip

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_clip(
HIG_PDE_ELEMENT hElement,
LPHIG_PDE_CLIP lphClip );
```

Arguments:

hElement	An element whose clip is obtained. **NOTE:** A clip may be shared by many elements. Use care when modifying a clip. Copy it first if you want to modify the clip for a specific element.
lphClip	Clip object for `hElement`.

Description:

Gets the current clip for an element.

Return Value:

Error count.

IG_PDE_element_set_clip

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_set_clip(
HIG_PDE_ELEMENT hElement,
HIG_PDE_CLIP hClip );
```

Arguments:

hElement	An element whose clip is set.
lphClip	The clip to set for `hElement`.

Description:

Sets the clip for an element.

Return Value:

Error count.

IG_PDE_element_get_gstate

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_gstate(
HIG_PDE_ELEMENT hElement,
LPAT_PDE_GRAPHICSTATE lpGstate );
```

Arguments:

hElement	An element whose graphics state is obtained.
lpGstate	Pointer to `AT_PDE_GRAPHICSTATE` structure that contains graphics state information for `hElement`. This graphics state information may contain PDE objects for color spaces or an `ExtGState`. They are not acquired by this method. **NOTE:** For a PDE Image, only the `ExtGState` value is used for images. For indexed images, the fill color space and values are categorized in the PDE Image object.

Description:

Gets the graphics state information for an element.

This method is only valid for PDE Form, PDE Image, PDE Path,and PDE Shading elements.

NOTE: Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this method. Be sure to release these non-NULL objects when disposing of lpGstate.

Return Value:

Error count.

IG_PDE_element_set_gstate

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_set_gstate(
HIG_PDE_ELEMENT hElement,
LPAT_PDE_GRAPHICSTATE lpGstate );
```

Arguments:

hElement	An element whose graphics state is set.
lpGstate	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state information to set for hElement.

Description:

Sets the graphics state information for an element.

This method is valid only for PDE Form, PDE Image, PDE Path, and PDE Shading elements.

Return Value:

Error count.

IG_PDE_element_get_matrix

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_matrix(
HIG_PDE_ELEMENT hElement,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hElement	An element whose transformation matrix is obtained.
lpMatrix	Pointer to AT_PDF_FIXEDMATRIX that holds a transformation matrix for hElement. If hElement is a text object, returns the identity matrix.

Description:

Gets the transformation matrix for an element.

This matrix provides the transformation from user space to device space for the element. If there is no cm operator (concatmatrix) in the page stream, the matrix is the identity matrix.

Return Value:

Error count.

IG_PDE_element_set_matrix

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_set_matrix(
HIG_PDE_ELEMENT hElement,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hElement	An element whose transformation matrix is set.
lpMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the transformation matrix to set for `hElement`.

Description:

Sets the transformation matrix for an element.

The element may not be a PDE Container, a PDE Group, a PDE Place, or a PDE Text.

Return Value:

Error count.

IG_PDE_element_get_dictionary

Group: PDE Element

Declaration:

```
IG_PDE_element_get_dictionary(
   HIG_PDE_ELEMENT hElement,
   LPHIG_PDF_DICTIONARY lphOCMD)
```

Arguments:

hElement	The Source Element.
lphOCMD	The returned Dictionary (OCMD object).

Description:

Returns a Dictionary (OCMD object), which is associated with this Element. If no Dictionary is associated, returns NULL.

Return Value:

Associated Dictionary, or NULL if no Dictionary is associated.

IG_PDE_element_get_unique_id

Group: PDE Object

Declaration:

```
IG_PDE_element_get_unique_id(
   HIG_PDE_ELEMENT hElement,
   LPLONG lpnId);
```

Arguments:

hElement	Element object.
lpnId	The unique identifier.

Description:

Returns some 32bit integer that is unique for all Element objects. In other words, it is guaranteed that there cannot be two Element objects with the same UniqueIds.

Can be used for Element objects' identification.

Return Value:

An unique identifier of this Element Object.

IG_PDE_element_has_gstate

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_has_gstate(
HIG_PDE_ELEMENT hElement,
LPAT_PDF_BOOL lpbHasGstate );
```

Arguments:

hElement	An element whose graphics state is checked.
lpbHasGstate	Returns True if the element has a graphics state; False otherwise.

Description:

Tests if `hElement` has a graphics state information.

Return Value:

Error count.

IG_PDE_element_release

Group: PDE Element

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_release(
HIG_PDE_ELEMENT hElement );
```

Arguments:

hElement	The element to release.

Description:

Release the specified element.

Return Value:

Error count.

HIG_PDE_CLIP

Type: PDE Clip Object

Description:

Handle to the PDE clip object. Clip – a list of elements containing a list of Paths and Texts that describe a clip state. Clips can be created and built up with PDE Clip methods. Any PDE Element object can have Clip associated with it. Clip objects can contain PDE Containers and PDE Groups to an arbitrary level of nesting. This allows PDE Containers to be used to mark clip objects.

PDE Groups inside PDE Clips that contain at least one PDE Text and no PDE Paths have a special meaning. All PDE Text objects contained in such a PDE Group are considered to be part of the same BT/ET block. This means that the union of these

PDE Texts makes up a single clipping path—as opposed to the intersection of the PDE Texts.

See Section 5.3 in the PDF reference for more information about BT/ET block.

Class Members:

IG_PDE_clip_create	Creates an empty clip object.
IG_PDE_clip_clone	Makes a deep copy of a PDE Clip object.
IG_PDE_clip_add_element	Adds an element to a clip path.
IG_PDE_clip_get_element	Gets an element from a clip object.
IG_PDE_clip_remove_elements	Removes one or more elements from a clip object.
IG_PDE_clip_get_element_count	Gets the number of top-level elements in a clip object.
IG_PDE_clip_enumerate_elements	For a given PDE Clip, enumerates all of the PDE Elements in a flattened manner.

IG_PDE_clip_create

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_create(
LPHIG_PDE_CLIP lphClip );
```

Arguments:

lphClip	The newly created clip object.

Description:

Creates an empty clip object. This represents a clipping object that has no effect on elements that refer to it.

Call `IG_PDE_element_release` to dispose the object.

Return Value:

Error count.

IG_PDE_clip_clone

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_clone(
HIG_PDE_CLIP hClip,
LPHIG_PDE_CLIP lphCloneClip);
```

Arguments:

hClip	The clipping path to copy.
lphCloneClip	The deep copy of `hClip`.

Description:

Makes a deep copy of a PDE Clip object.

Return Value:

Error count.

IG_PDE_clip_add_element

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_add_element(
HIG_PDE_CLIP hClip,
LONG nAfterIndex,
HIG_PDE_ELEMENT hElement);
```

Arguments:

hClip	The clip path to which an element is added.
nAfterIndex	The index after which to add `hElement`. Use `IG_PDE_BEFORE_FIRST` to insert an element at the beginning of the clip object.
hElement	The element added, which may be a PDE Path, a PDE Text, a PDE Container, a PDE Group or a PDE Place object.

Description:

Adds an element to a clip path.

Return Value:

Error count.

IG_PDE_clip_get_element

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_get_element(
HIG_PDE_CLIP hClip,
LONG nIndex,
LPHIG_PDE_ELEMENT lphElement);
```

Arguments:

hClip	The clip object from which an element is obtained.
nIndex	Index of element to get from clip.
lphElement	The element from the clip object

Description:

Gets an element from a clip object.

Return Value:

Error count.

IG_PDE_clip_remove_elements

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_remove_elements(
HIG_PDE_CLIP hClip,
LONG nIndex,
LONG nCount );
```

Arguments:

hClip	The clip object from which an element is removed.
nIndex	First element to remove.
nCount	Number of elements to remove.

Description:

Removes one or more elements from a clip object.

Return Value:

Error count.

IG_PDE_clip_get_element_count

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_get_element_count(
HIG_PDE_CLIP hClip,
LPUINT lpnCount);
```

Arguments:

hClip	The clip object to examine.
lpnCount	Number of path and charpath elements in clip. If clip contains PDE Groups, this method returns the top-level PDE Path, PDE Text, PDE Container, PDE Group, or PDE Place object. Use `IG_PDE_clip_enumerate_elements` to see only the PDE Path and PDE Text objects. **NOTE:** PDEGroup is not a persistent object. You cannot save to PDF and re-get group objects.

Description:

Gets the number of top-level elements in a clip object. Top-level elements may be a path or char-path, a marked content container or place, or a group.

Paths are represented as PDE Path objects; char-paths are represented as PDE Text objects.

Return Value:

Error count.

IG_PDE_clip_enumerate_elements

Group: PDE Clip

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_enumerate_elements (
HIG_PDE_CLIP hClip,
LPFNIG_PDE_CLIP_ENUMPROC lpfnEnumProc,
LPVOID lpClientData,
LPAT_PDF_BOOL lpbResult );
```

Arguments:

hClip	The PDEClip to enumerate.
lpfnEnumProc	Called with each flattened element. Enumeration continues until all elements have been enumerated, or until `lpfnEnumProc` returns false.
lpClientData	Pointer to user-supplied data to pass to `lpfnEnumProc` each time it is called.
lpbResult	Returns value of `lpfnEnumProc`. True if successful, False otherwise.

Description:

For a given PDE Clip, enumerates all of the PDE Elements in a flattened manner. In other words, PDE Containers and PDE Groups nested in the PDE Clip will not be handed back, but any PDE Paths and PDE Texts nested in them will be. Additionally, PDE Place objects inside the PDE Clip are not returned.

Return Value:

Error count.

HIG_PDE_CONTAINER

Type: PDE Container Object

Description:

Handle to the PDE container object. Container - a group of elements on a page in a PDF file. In the PDF file, containers are delimited by Marked Content BMC/EMC or BDC/EMC pairs. Every container has a Marked Content tag associated with it. In addition to grouping a set of elements, a BDC/EMC pair specifies a property list to be associated with the grouping. Thus a container corresponding to a BDC/EMC pair also has a property list dictionary associated with it.

See Section 10.5 in the PDF reference for more information about Marked Content operators.

Class Members:

IG_PDE_container_create	Creates a container object.
IG_PDE_container_get_content	Gets the PDE Content for `hContainer`.
IG_PDE_container_set_content	Sets the content for a container.
IG_PDE_container_get_dictionary	Gets the Marked Content dictionary for a container.
IG_PDE_container_set_dictionary	Changes the Marked Content dictionary for a container.
IG_PDE_container_get_mctag	Gets the Marked Content tag for a container.
IG_PDE_container_set_mctag	Sets the Marked Content tag for a `hContainer`.

IG_PDE_container_create

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_create(
HIG_PDF_ATOM hMCTag,
HIG_PDF_BASOBJ hDictionary,
AT_PDF_BOOL bIsInline,
LPHIG_PDE_CONTAINER lphContainer );
```

Arguments:

hMCTag	Tag name for the container.
hDictionary	Optional Marked Content dictionary for the container.
bIsInline	If True, emits container into the page content stream inline.
lphContainer	The newly created container object.

Description:

Creates a container object.

Call `IG_PDE_element_release` to dispose the object.

Return Value:

Error count.

IG_PDE_container_get_content

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_get_content(
HIG_PDE_CONTAINER hContainer,
LPHIG_PDE_CONTENT lphContent );
```

Arguments:

hContainer	A container whose content is obtained.
lphContent	The PDE Content for `hContainer`.

Description:

Gets the PDE Content for `hContainer`.

Return Value:

Error count.

IG_PDE_container_set_content

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_set_content(
```

```
HIG_PDE_CONTAINER hContainer,
HIG_PDE_CONTENT hContent );
```

Arguments:

hContainer	A container whose content is set.
hContent	The content of `hContainer`.

Description:

Sets the content for a container. The existing PDE Content is released by this method.

Return Value:

Error count.

IG_PDE_container_get_dictionary

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_get_dictionary(
HIG_PDE_CONTAINER hContainer,
LPHIG_PDF_BASOBJ lphDictionary,
LPAT_PDF_BOOL lpbIsInline,
LPAT_PDF_BOOL lpbResult );
```

Arguments:

hContainer	A container.
lphDictionary	Marked Content dictionary for `hContainer`. NULL if `hContainer` has no Marked Content dictionary.
lpbIsInline	True if the dictionary is inline, false otherwise. Undefined if `hContainer` has no Marked Content dictionary.
lpbResult	True if `hContainer` has a Marked Content dictionary, False otherwise.

Description:

Gets the Marked Content dictionary for a container.

Return Value:

Error count.

IG_PDE_container_set_dictionary

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_set_dictionary(
HIG_PDE_CONTAINER hContainer,
HIG_PDF_BASOBJ hDictionary,
AT_PDF_BOOL bIsInline );
```

Arguments:

hContainer	A container whose dictionary is changed.
hDictionary	Marked Content dictionary being set into `hContainer`.
bIsInline	If True, the dictionary is emitted inline.

Description:

Changes the Marked Content dictionary for a container.

Return Value:

Error count.

IG_PDE_container_get_mctag

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_get_mctag (
HIG_PDE_CONTAINER hContainer,
LPHIG_PDF_ATOM lphMCTag );
```

Arguments:

hContainer	A container.
lphMCTag	Marked Content tag of `hContainer`. Returns `IG_PDF_ATOM_NULL` if `hContainer` has no Marked Content tag.

Description:

Gets the Marked Content tag for a container.

Return Value:

Error count.

IG_PDE_container_set_mctag

Group: PDE Container

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_set_mctag (
HIG_PDE_CONTAINER hContainer,
HIG_PDF_ATOM hMCTag );
```

Arguments:

hContainer	A container to tag.
hMCTag	Marked Content tag.

Description:

Sets the Marked Content tag for a `hContainer`.

Return Value:

Error count.

HIG_PDE_FORM

Type: PDE Form Object

Description:

Handle to the PDE form object. Form - an element that corresponds to an instance of XObject Form on a page (or other containing stream such as another XObject Form or annotation form). The context associated with this instance includes the actual stream that represents the XObject Form and the initial conditions of the graphics state. The latter consists of the transformation matrix, initial color values, and so forth. It is possible to have two Forms that refer to the same XObject Form. The forms will exist at different places on the same page, depending on the transformation matrix. They may also have different colors or line stroking parameters. In the case of a transparency group, the opacity is specified in the gstate.

Within a Form, each element has its own gstate (or is a container, place, or group object). These gstates are independent of the parent Form gstate. Form elements may have their own opacity.

Content may be obtained from a Form to edit the form's display list.

Class Members:

IG_PDE_form_create	Creates a new form from an existing object.
IG_PDE_form_clone	Creates a new form from an existing form object.
IG_PDE_form_get_content	Gets a PDE Content object for a form.
IG_PDE_form_set_content	Sets the form content.
IG_PDE_form_has_xgroup	Determines whether the XObject form has a Transparency XGroup
IG_PDE_form_get_xgroup	Acquires the transparency group dictionary of the XObject form.
IG_PDE_form_set_xgroup	Sets the transparency group dictionary of the form XObject.

IG_PDE_form_create

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_create (
HIG_PDF_BASOBJ hXObject,
HIG_PDF_BASOBJ hResources,
LPAT_PDF_FIXEDMATRIX lpMatrix,
```

```
LPHIG_PDE_FORM lphForm );
```

Arguments:

hXObject	XObject from which a form is created.
hResources	The hXObject's Resources dictionary. If you do not pass in a Resource object, subsequent calls to `IG_PDF_page_get_content` will fail (after the file is saved).
lpMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the transformation matrix to use for the form.
lphForm	The newly created form object.

Description:

Creates a new form from an existing object.

Call `IG_PDE_element_release` to dispose the object.

Return Value:

Error count.

IG_PDE_form_clone

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_clone (
HIG_PDE_FORM hForm,
LPHIG_PDE_FORM lphCloneForm );
```

Arguments:

hForm	Form object from which a new PDE Form is created.
lphCloneForm	The newly created form object.

Description:

Creates a new form from an existing form object. Creates a copy of the PDE Form, including the underlying objects.

Return Value:

Error count.

IG_PDE_form_get_content

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_get_content(
HIG_PDE_FORM hForm,
LPHIG_PDE_CONTENT lphContent );
```

Arguments:

hForm	The form whose content is obtained.
lphContent	Form content object.

Description:

Gets a PDE Content object for a form.

Return Value:

Error count.

IG_PDE_form_set_content

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_set_content(
   HIG_PDE_FORM hForm,
   HIG_PDE_CONTENT hContent );
```

Arguments:

hForm	The form whose content is set.
hContent	The new content for form.

Description:

Sets the form content.

Return Value:

Error count.

IG_PDE_form_has_xgroup

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_has_xgroup(
HIG_PDE_FORM hForm,
LPAT_PDF_BOOL lpbHasXGroup );
```

Arguments:

hForm	The form object.
lpbHasXGroup	True if the XObject form has a Transparency XGroup.

Description:

Determines whether the XObject form has a Transparency XGroup

Return Value:

Error count.

IG_PDE_form_get_xgroup

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_get_xgroup(
HIG_PDE_FORM hForm,
LPHIG_PDE_XGROUP lphXGroup );
```

Arguments:

hForm	The form whose XGroup is obtained.
lphXGroup	Transparency group object.

Description:

Acquires the transparency group dictionary of the XObject form.

Return Value:

Error count.

IG_PDE_form_set_xgroup

Group: PDE Form

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_set_xgroup(
HIG_PDE_FORM hForm,
HIG_PDE_XGROUP hXGroup );
```

Arguments:

hForm	The form whose XGroup is set.
hXGroup	The transparency dictionary.

Description:

Sets the transparency group dictionary of the form XObject.

Return Value:

Error count.

HIG_PDE_GROUP

Type: PDE Group Object

Description:

Handle to the PDE group object. Group - an in-memory representation of objects in Content. It has no state and is not represented in any way in a content stream (that is, Content).

When used in a Clip, this object is used to associate Text objects into a single clipping object.

Class Members:

IG_PDE_group_create	Creates a PDE Group object.
IG_PDE_group_get_content	Gets a PDE Content object for a group.
IG_PDE_group_set_content	Sets the group's content.

IG_PDE_group_create

Group: PDE Group

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_group_create (
LPHIG_PDE_GROUP lphGroup );
```

Arguments:

lphGroup	The newly created group object.

Description:

Creates a PDE Group object.

Return Value:

Error count.

IG_PDE_group_get_content

Group: PDE Group

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_group_get_content(
HIG_PDE_GROUP hGroup,
LPHIG_PDE_CONTENT lphContent );
```

Arguments:

hGroup	The group whose content is obtained.
lphContent	Group content object.

Description:

Gets a PDE Content object for a group.

Return Value:

Error count.

IG_PDE_group_set_content

Group: PDE Group

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_group_set_content(
HIG_PDE_GROUP hGroup,
HIG_PDE_CONTENT hContent );
```

Arguments:

hGroup	The group whose content is set.
hContent	The new content for group.

Description:

Sets the group's content.

Return Value:

Error count.

HIG_PDE_IMAGE

Type: PDE Image Object

Description:

Handle to the PDE image object. Image - an element that contains an Image XObject or in-line image. You can associate data or a stream with an image.

Class Members:

IG_PDE_image_create	Creates an image object.
IG_PDE_image_is_data_encoded	Determines if image data is encoded or not.
IG_PDE_image_get_attrs	Gets the attributes for an image.
IG_PDE_image_get_color_mask	Gets the Mask entry from the image dictionary.
IG_PDE_image_get_colorspace	Gets the color space object for an image.
IG_PDE_image_get_data	Gets an image's data.
IG_PDE_image_get_data_length	Gets the length of data for an image.
IG_PDE_image_get_data_stream	Gets a data stream for an image.
IG_PDE_image_get_decode_array	Gets the decode array for an image.
IG_PDE_image_get_filter_array	Gets the filter array for an image.
IG_PDE_image_get_matte_array	Gets the matte array for the image XObject.
IG_PDE_image_get_soft_mask	Gets the soft mask for an image.
IG_PDE_image_has_soft_mask	Checks whether the image has a soft mask.
IG_PDE_image_is_xobject	Determines if an image is an XObject image.
IG_PDE_image_set_color_mask	Sets the color space of the image.
IG_PDE_image_set_colorspace	Sets the Mask entry from the image dictionary.
IG_PDE_image_set_data	Sets data for an image.
IG_PDE_image_set_data_stream	Sets a data stream for an image.
IG_PDE_image_set_decode_array	Sets the decode array of an image.
IG_PDE_image_set_matte_array	Sets the matte array for the image XObject.
IG_PDE_image_set_soft_mask	Sets the soft mask.

IG_PDE_image_create

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_create(
LPAT_PDE_IMAGEATTRS lpAttrs,
LPAT_PDF_FIXEDMATRIX lpMatrix,
LONG nFlag,
HIG_PDE_COLORSPACE hColorSpace,
LPAT_PDE_COLORVALUE lpColorValue,
```

```
    LPAT_PDE_FILTERARRAY lpFilters,
    HIG_PDF_STREAM hDataStream,
    LPBYTE lpData,
    LONG nDataLen,
    LPHIG_PDE_IMAGE lphImage );
```

Arguments:

lpAttrs	Pointer to `AT_PDE_IMAGEATTRS` with attributes of the image.
lpMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the transformation matrix to use for the image.
nFlag	`enumIGPDEImageDataFlags` flags. If the `AT_PDE_IMAGE_ENCODED_DATA` flag is set, and the data is provided directly (not as a stream), then `nDataLen` must specify the length of data.
hColorSpace	Color space of the image. When the image is an imagemask, `hColorSpace` is the color space of the `lpColorValue` argument.
lpColorValue	Pointer to `AT_PDE_COLORVALUE` structure. If the image is an image mask, `lpColorValue` must be provided.
lpFilters	Pointer to `AT_PDE_FILTERARRAY` structure that specifies which filters to use in encoding the contents; may be NULL. Filters will be used to encode the data in the order in which they are specified in the array.
hDataStream	Stream holding the image data.
lpData	Image data. If `hDataStream` is non-NULL, data is ignored. If there is a great deal of data, as for a large image, it is recommended you use the `hDataStream` parameter for the image data.
nDataLen	Encoded length of `lpData`, in bytes.
lphImage	The image object.

Description:

Creates an image object.

The image data may be specified as a stream or as a buffer. If `hDataStream` is non-NULL, `lpData` is ignored.

See `IG_PDE_image_set_data_stream` for information on handling the stream.

The caller must dispose of `hDataStream` after calling this method.

Call `IG_PDE_element_release` to dispose the created image object when finished with it.

Return Value:

Error count.

IG_PDE_image_is_data_encoded

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_is_data_encoded(
HIG_PDE_IMAGE hImage,
LPAT_PDF_BOOL lpbIsEncoded,
LPDWORD lpnEncodedLen );
```

Arguments:

hImage	Image to examine.
lpbIsEncoded	True if `IG_PDE_image_get_data` returns encoded data, false otherwise. Returns false for XObject images.
lpnEncodedLen	Length of the encoded data—if the data is encoded, that is, `lpbIsEncoded` returns true.

Description:

Determines if image data is encoded or not. Used only for inline images; not relevant to XObject images.

`lpbIsEncoded` always returns false for XObject images; XObject image data can be obtained from `IG_PDE_image_get_data` or `IG_PDE_image_get_data_stream`, either encoded or decoded.

Only if `IG_PDE_image_create` is used to explicitly create a new image using encoded data does `lpbIsEncoded` returns true.

Return Value:

Error count.

IG_PDE_image_get_attrs

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_attrs (
HIG_PDE_IMAGE hImage,
LPAT_PDE_IMAGEATTRS lpAttrs );
```

Arguments:

hImage	Image whose attributes are obtained.
lpAttrs	Pointer to AT_PDE_IMAGEATTRS structure with attributes of image.

Description:

Gets the attributes for an image.

Return Value:

Error count.

IG_PDE_image_get_color_mask

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_color_mask(
   HIG_PDE_IMAGE hImage,
   LPLONG lpMask,
   LPUINT lpnLen
   );
```

Arguments:

hImage	The image object whose color mask is obtained.
lpMask	A pointer to the array of LONG values to fill with color mask values. lpMask must contain enough values to hold the entire color mask array.
lpnLen	The number of color mask elements obtained by the method - size of lpMask array.

Description:

Use this method to obtain the Mask entry from the image dictionary. The mask entry is an array specifying a range of colors to be masked out. Samples in the image that fall within this range are not painted, allowing the existing background to show through. The effect is similar to that of the video technique known as chroma-key.

The value of each Mask entry is an array of 2n integers, [min1 max1 ... minn maxn], where n is the number of color components in the image's color space. Each integer must be in the range 0 to (2^BitsPerComponent - 1), representing color values before decoding with the Decode array. An image sample is masked (not painted) if all of its color components before decoding, c1…cn, fall within the specified ranges.

Return Value:

Error count.

IG_PDE_image_get_colorspace

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_colorspace (
HIG_PDE_IMAGE hImage,
LPHIG_PDE_COLORSPACE lphColorSpace );
```

Arguments:

hImage	Image whose colorspace is obtained.
lphColorSpace	Color space for `hImage`. Returns NULL if `hImage` is an image mask.

Description:

Gets the color space object for an image.

Return Value:

Error count.

IG_PDE_image_get_data

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_data (
HIG_PDE_IMAGE hImage,
LONG nFlags,
LPBYTE lpData );
```

Arguments:

hImage	Image whose data is obtained.
nFlags	Unused, must be 0.
lpData	Image data. If the data is decoded, `lpData` must be large enough to contain the number of bytes specified in the `AT_PDE_IMAGEATTRS` structure obtained by `IG_PDE_image_get_attrs`. If the data is encoded, `lpData` must be large enough to contain the number of bytes in the `lpnEncodedLen` parameter obtained by `IG_PDE_image_is_data_encoded`.

Description:

Gets an image's data.

If the image is a XObject image, data is always returned as decoded data.

See the note about inline images under `IG_PDE_image_is_data_encoded`.

Return Value:

Error count.

IG_PDE_image_get_data_length

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_data_length (
HIG_PDE_IMAGE hImage,
LPLONG lpnLength );
```

Arguments:

hImage	Image whose data length is obtained.
lpnLength	Number of bytes of image data, specified by the width, height, bits per component, and color space of the image.

Description:

Gets the length of data for an image.

Return Value:

Error count.

IG_PDE_image_get_data_stream

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_data_stream (
HIG_PDE_IMAGE hImage,
LONG nFlags,
LPHIG_PDF_STREAM lphStream );
```

Arguments:

hImage	Image whose data stream is obtained.
nFlags	`enumIGPDEImageDataFlags` flags. If the `AT_PDE_IMAGE_ENCODED_DATA` flag is set, data is returned in encoded form. Otherwise, data is decoded.
lphStream	Stream for `hImage`.

Description:

Gets a data stream for an image. May only be called for XObject images.

The caller must dispose of the returned stream by calling `IG_PDF_stream_close`.

Return Value:

Error count.

IG_PDE_image_get_decode_array

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_decode_array (
HIG_PDE_IMAGE hImage,
LPAT_PDF_FIXED lpDecode,
LONG nDecodeSize,
LPLONG lpnLength );
```

Arguments:

hImage	The image whose decode array is obtained.
lpDecode	Pointer to the decode array. If NULL, the number of decode elements required is returned via `lpnLength`.
nDecodeSize	Size of `lpDecode` in bytes.
lpnLength	Number of elements in the decode array.

Description:

Gets the decode array for an image.

Return Value:

Error count.

IG_PDE_image_get_filter_array

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_filter_array (
HIG_PDE_IMAGE hImage,
LPAT_PDE_FILTERARRAY lpFilters
LPLONG lpnLength );
```

Arguments:

hImage	Image whose filter array is obtained.
lpFilters	Pointer to AT_PDE_FILTERARRAY structure to fill with the current filter array for the image. lpFilters must be large enough to contain all of the elements. May be NULL to obtain the number of filter elements via lpnLength.
lpnLength	Number of filter elements.

Description:

Gets the filter array for an image.

Return Value:

Error count.

IG_PDE_image_get_matte_array

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_matte_array (
HIG_PDE_IMAGE hImage,
LPAT_PDF_FIXED lpMatte,
LONG nValuesCount,
LPLONG lpnLength );
```

Arguments:

hImage	The image XObject.
lpMatte	An array of values.
nValuesCount	The number of values in `lpMatte`.
lpnLength	Number of values copied.

Description:

Gets the matte array for the image XObject.

Return Value:

Error count.

IG_PDE_image_get_soft_mask

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_soft_mask (
HIG_PDE_IMAGE hImage,
LPHIG_PDE_IMAGE lphSoftMask );
```

Arguments:

hImage	The image object.
lphSoftMask	The soft mask for image.

Description:

Gets the soft mask for an image.

Use `IG_PDE_element_release` to dispose of the object when it is no longer referenced.

Return Value:

Error count.

IG_PDE_image_has_soft_mask

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_has_soft_mask (
HIG_PDE_IMAGE hImage,
LPAT_PDF_BOOL lpbHasSoftMask );
```

Arguments:

hImage	The image object.
lpbHasSoftMask	True if the soft mask exists; False otherwise.

Description:

Checks whether the image has a soft mask.

Return Value:

Error count.

IG_PDE_image_is_xobject

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_is_xobject (
HIG_PDE_IMAGE hImage,
LPAT_PDF_BOOL lpbIsXObject );
```

Arguments:

hImage	The image object.
lpbIsXObject	Trrue if the image is an XObject image, false otherwise.

Description:

Determines if an image is an XObject image.

Return Value:

Error count.

IG_PDE_image_set_color_mask

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_color_mask(
   HIG_PDE_IMAGE hImage,
   LPLONG lpMask,
   UINT nLen
   );
```

Arguments:

hImage	The image object whose color mask is set.
lpMask	A pointer to the array of LONG values containing the color mask values.
nLen	The number of color mask elements in lpMask - size of lpMask array.

Description:

Use this method to set the Mask entry from the image dictionary. The mask entry is an array specifying a range of colors to be masked out. Samples in the image that fall within this range are not painted, allowing the existing background to show through. The effect is similar to that of the video technique known as chroma-key.

The value of each Mask entry is an array of 2n integers, [min1 max1 ... minn maxn], where n is the number of color components in the image's color space. Each integer must be in the range 0 to (2^BitsPerComponent - 1), representing color values before decoding with the Decode array. An image sample is masked (not painted) if all of its color components before decoding, c1…cn, fall within the specified ranges.

Return Value:

Error count.

Example:

```
//24-bit rgb color mask with 1 mask entry
LONG lpMask[] = {250, 255, 250, 255, 250, 255};
int iMaskLen=6;
IG_PDE_image_set_color_mask(hElement,lpMask,iMaskLen);
```

IG_PDE_image_set_colorspace

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_colorspace (
HIG_PDE_IMAGE hImage,
HIG_PDE_COLORSPACE hColorSpace );
```

Arguments:

hImage	Image whose colorspace is set.
hColorSpace	PDE ColorSpace object.

Description:

Sets the color space of the image.

Return Value:

Error count.

IG_PDE_image_set_data

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_data (
HIG_PDE_IMAGE hImage,
LONG nFlags,
LPBYTE lpData,
LONG nLength );
```

Arguments:

hImage	Image whose data is set.
nFlags	A set of `enumIGPDEImageDataFlags` flags. If `AT_PDE_IMAGE_ENCODED_DATA` is set, the data must be encoded for the current filters, and nLength is the length of the encoded data. If the `AT_PDE_IMAGE_ENCODED_DATA` flag is not set, data is not encoded and `nLength` is the size of the decoded data.
lpData	Image data.
nLength	Length of data.

Description:

Sets data for an image.

Return Value:

Error count.

IG_PDE_image_set_data_stream

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_data_stream (
HIG_PDE_IMAGE hImage,
LONG nFlags,
LPAT_PDE_FILTERARRAY lpFilters,
HIG_PDF_STREAM hDataStream );
```

Arguments:

hImage	Image whose data stream is set.
nFlags	`enumIGPDEImageDataFlags` flags. If the `AT_PDE_IMAGE_ENCODED_DATA` flag is set, the stream must be encoded.
lpFilters	Pointer to `AT_PDE_FILTERARRAY` structure. If not NULL, is used to build the objects for the Filter, DecodeParms, and EncodeParms objects. If `lpFilters` is NULL, the existing Filter and DecodeParms are used. EncodeParms is set to DecodeParms if it exists.
hDataStream	Stream for the image data.

Description:

Sets a data stream for an image. Can only be used for XObject images.

Return Value:

Error count.

IG_PDE_image_set_decode_array

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_decode_array (
HIG_PDE_IMAGE hImage,
LPAT_PDF_FIXED lpDecode,
LONG nDecodeSize );
```

Arguments:

hImage	Image whose decode array is set.
lpDecode	Pointer to the decode array.
nDecodeSize	Size of decode array in bytes.

Description:

Sets the decode array of an image.

Normally, the decode array is accessed through the decode field in the `AT_PDE_IMAGEATTRS` structure. However, this method defines a decode array to handle images with a color space that has more than 4 components.

Return Value:

Error count.

IG_PDE_image_set_matte_array

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_matte_array (
HIG_PDE_IMAGE hImage,
LPAT_PDF_FIXED lpMatte,
LONG nValuesCount );
```

Arguments:

hImage	The image XObject.
lpMatte	An array of values.
nValuesCount	The number of values in `lpMatte`.

Description:

Sets the matte array for the image XObject.

Return Value:

Error count.

IG_PDE_image_set_soft_mask

Group: PDE Image

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_soft_mask (
HIG_PDE_IMAGE hImage,
HIG_PDE_IMAGE hSoftMask );
```

Arguments:

hImage	The image XObject.
hSoftMask	The soft mask.

Description:

Sets the soft mask.

Return Value:

Error count.

HIG_PDE_PATH

Type: PDE Path Object

Description:

Handle to the PDE path object./ Path – an element that contains a path. Path objects can be stroked, filled, and/or serve as a clipping path.

Class Members:

IG_PDE_path_create	Creates an empty path element.
IG_PDE_path_add_segment	Adds a segment to a path.
IG_PDE_path_get_data	Gets the size of the path data and, optionally, the path data.
IG_PDE_path_set_data	Sets new path data for a path element.
IG_PDE_path_get_paint_op	Gets the fill and stroke attributes of a path.
IG_PDE_path_set_paint_op	Sets the fill and stroke attributes of a path.

IG_PDE_path_create

Group: PDE Path

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_create(
LPHIG_PDE_PATH lphPath );
```

Arguments:

lphPath	Newly created empty path element.

Description:

Creates an empty path element.

Call `IG_PDE_element_release` to dispose the created path object when finished with it.

Return Value:

Error count.

IG_PDE_path_add_segment

Group: PDE Path

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_add_segment(
HIG_PDE_PATH hPath,
LONG nSegType,
AT_PDF_FIXED x1,
AT_PDF_FIXED y1,
AT_PDF_FIXED x2,
AT_PDF_FIXED y2,
AT_PDF_FIXED x3,
AT_PDF_FIXED y3 );
```

Arguments:

hPath	The path to which a segment is added.
nSegType	A `enumIGPDEPathElementType` value indicating the type of path to add.
x1	x-coordinate of first point.
y1	y-coordinate of first point.
x2	x-coordinate of second point.
y2	y-coordinate of second point.
x3	x-coordinate of third point.
y3	y-coordinate of third point.

Description:

Adds a segment to a path. The number of `AT_PDF_FIXED` values used depends upon `nSegType`:

`IG_PDE_MOVE_TO`: x1, y1

`IG_PDE_LINE_TO`: x1, y1

`IG_PDE_CURVE_TO`: x1, y1, x2, y2, x3, y3

`IG_PDE_CURVE_TO_V`: x1, y1, x2, y2

`IG_PDE_CURVE_TO_Y`: x1, y1, x2, y2

`IG_PDE_RECT`: x1, y1, x2 (width), y2 (height)

`IG_PDE_CLOSE_PATH`: None

Return Value:

Error count.

IG_PDE_path_get_data

Group: PDE Path

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_get_data(
    HIG_PDE_PATH hPath,
    LPLONG lpData,
    LONG nDataSize,
    LPLONG lpnLenght  );
```

Arguments:

hPath	The path whose data is obtained.
lpData	Pointer to path data. If `lpData` is non-NULL, it contains a variable-sized array of path operators and operands. The format is a 32-bit operator followed by 0 to 3 `AT_PDF_FIXEDPOINT` values, depending on the operator. Opcodes are codes for moveto, lineto, curveto, rect, or closepath operators; operands are `AT_PDF_FIXEDPOINT` values. If data is NULL, the number of bytes required for data is returned in `lpnLenght`. **NOTE:** Returns “raw” path data. If you want the points in page coordinates, concatenate the path data points with the PDE Element matrix obtained from `IG_PDE_element_get_matrix`.
nDataSize	Specifies the size of the buffer provided in data. If it is less than the length of the path data, the method copies datasize bytes.
lpnLength	Length of data of `hPath`.

Description:

Gets the size of the path data and, optionally, the path data.

Return Value:

Error count.

IG_PDE_path_set_data

Group: PDE Path

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_set_data(
HIG_PDE_PATH hPath,
LPLONG lpData,
LONG nDataSize );
```

Arguments:

hPath	The path whose data is set.
lpData	Pointer to path data. It is a variable-sized array of path operators and operands. The format is a 32-bit operator followed by 0 to 3 `AT_PDF_FIXEDPOINT` values, depending on the operator. Operators are codes for moveto, lineto, curveto, rect, or closepath operators and must be one of `enumIGPDEPathElementType`. Operands are `AT_PDF_FIXEDPOINT` values. The data is copied into hPath object.
nDataSize	Size of the new path data, in bytes.

Description:

Sets new path data for a path element.

Return Value:

Error count.

IG_PDE_path_get_paint_op

Group: PDE Path

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_get_paint_op(
HIG_PDE_PATH hPath,
LPLONG lpnPaintOpAttrs );
```

Arguments:

hPath	The path whose fill and stroke attributes are obtained.
lpnPaintOpAttrs	A set of `enumIGPDEPathOpFlags` flags describing fill and stroke attributes.

Description:

Gets the fill and stroke attributes of a path.

Return Value:

Error count.

IG_PDE_path_set_paint_op

Group: PDE Path

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_set_paint_op(
HIG_PDE_PATH hPath,
LONG nPaintOpAttrs );
```

Arguments:

hPath	The path whose fill and stroke attributes are set.
nPaintOpAttrs	The operation to set; must be one of `enumIGPDEPathOpFlags`.

Description:

Sets the fill and stroke attributes of a path.

Return Value:

Error count.

HIG_PDE_PLACE

Type: PDE Place Object

Description:

Handle to the PDE place object. Place – an element that marks a place on a page in a PDF file. In a PDF file, a place is represented by the MP or DP Marked Content operators.

Marked content is useful for adding structure information to a PDF file. For instance, a drawing program may want to mark a point with information, such as the start of a path of a certain type. Marked content provides a way to retain this information in the PDF file. A

DP operator functions the same as the MP operator and, in addition, allows a property list dictionary to be associated with a place.

Class Members:

IG_PDE_place_create	Creates a place object.
IG_PDE_place_get_dictionary	Gets the Marked Content dictionary for `hPlace`.
IG_PDE_place_set_dictionary	Sets the Marked Content dictionary for `hPlace`.
IG_PDE_place_get_mctag	Gets the Marked Content tag for a `hPlace`.
IG_PDE_place_set_mctag	Sets the Marked Content tag for a `hPlace`.

IG_PDE_place_create

Group: PDE Place

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_create(
HIG_PDF_ATOM mcTag,
HIG_PDF_BASOBJ mcDict,
AT_PDF_BOOL bIsInline
LPHIG_PDE_PLACE lphPlace );
```

Arguments:

mcTag	Tag name for the place. Must not contain any white space characters (for example, spaces or tabs).
mcDict	Optional Marked Content dictionary associated with the place.
bIsInline	If True, place is emitted into the page content stream inline.
lphPlace	The place object

Description:

Creates a place object.

Call `IG_PDE_element_release` to dispose the created place object when finished with it.

Return Value:

Error count.

IG_PDE_place_get_dictionary

Group: PDE Place

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_get_dictionary(
HIG_PDE_PLACE hPlace,
LPHIG_PDF_BASOBJ lpmcDict,
LPAT_PDF_BOOL lpbIsInline,
LPAT_PDF_BOOL lpbResult );
```

Arguments:

hPlace	The place whose Marked Content dictionary is obtained.
lpmcDict	Pointer to the Marked Content dictionary; may be NULL.
lpbIsInline	If True, the Marked Content dictionary is inline; may be NULL.
lpbResult	True if dictionary is obtained, False if no dictionary is present.

Description:

Gets the Marked Content dictionary for `hPlace`.

Return Value:

Error count.

IG_PDE_place_set_dictionary

Group: PDE Place

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_set_dictionary(
HIG_PDE_PLACE hPlace,
HIG_PDF_BASOBJ mcDict,
AT_PDF_BOOL bIsInline );
```

Arguments:

hPlace	The place whose Marked Content dictionary is set.
lpmcDict	Marked Content dictionary for `hPlace`.
lpbIsInline	If True, the dictionary is emitted inline.

Description:

Sets the Marked Content dictionary for `hPlace`.

Return Value:

Error count.

IG_PDE_place_get_mctag

Group: PDE Place

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_get_mctag(
HIG_PDE_PLACE hPlace,
LPHIG_PDF_ATOM lpmcTag );
```

Arguments:

hPlace	The place whose Marked Content tag is obtained.
lpmcTag	Tag for `hPlace`.

Description:

Gets the Marked Content tag for a `hPlace`.

Return Value:

Error count.

IG_PDE_place_set_mctag

Group: PDE Place

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_set_mctag(
HIG_PDE_PLACE hPlace,
HIG_PDF_ATOM mcTag );
```

Arguments:

hPlace	The place whose Marked Content tag is set.
mcTag	The tag for `hPlace`.

Description:

Sets the Marked Content tag for a hPlace.

Return Value:

Error count.

HIG_PDE_POSTSCRIPT

Type: PDE PostScript Object

Description:

Handle to the PDE PostScript object. PostScript – an element representing in-line or XObject pass-through PostScript object. XObject PostScripts are listed in page XObject resources.

Class Members:

IG_PDE_postscript_create	Creates a PDE PostScript object.
IG_PDE_postscript_get_attrs	Gets `hPostScript` attributes.
IG_PDE_postscript_get_data	Gets all or part of the image data.
IG_PDE_postscript_set_data	Sets the data for `hPostScript`.
IG_PDE_postscript_get_data_stream	Gets a stream for the data.
IG_PDE_postscript_set_data_stream	Sets a stream for the data.

IG_PDE_postscript_create

Group: PDE PostScript

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_create(
LPAT_PDE_PSATTRS lpAttrs,
HIG_PDF_STREAM hDataStream,
LPBYTE lpData,
LONG nDataSize
LPHIG_PDE_POSTSCRIPT lphPostScript );
```

Arguments:

lpAttrs	Pointer to AT_PDE_PSATTRS attributes data structure.
hDataStream	Data stream. May be NULL.
lpData	Data. May be NULL.
nDataSize	Number of bytes of data.
lphPostScript	The postscript object

Description:

Creates a PDE PostScript object. `lpData` and `hDataStream` may be NULL. If so, use `IG_PDE_postscript_set_data` and `IG_PDE_postscript_set_data_stream` to attach data to the object. If `hDataStream` is non-NULL, then data will be ignored.

If data is non-NULL and `hDataStream` is NULL, the data must contain `nDataSize` number of bytes as specified in the `AT_PDE_PSATTRS`.

Call `IG_PDE_element_release` to dispose the created object when finished with it.

Return Value:

Error count.

IG_PDE_postscript_get_attrs

Group: PDE PostScript

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_get_attrs(
HIG_PDE_POSTSCRIPT hPostScript,
LPAT_PDE_PSATTRS lpAttrs);
```

Arguments:

hPostScript	PDE postscript object.
lpAttrs	Pointer to AT_PDE_PSATTRS data structure containing the attributes information.

Description:

Gets `hPostScript` attributes.

Return Value:

Error count.

IG_PDE_postscript_get_data

Group: PDE PostScript

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_get_data (
HIG_PDE_POSTSCRIPT hPostScript,
LPBYTE lpBuffer,
LONG nBufferSize,
LONG nOffset,
LPLONG lpnBytesWritten);
```

Arguments:

hPostScript	PDE postscript object.
lpBuffer	Receives the data.
nBufferSize	Size of the buffer.
nOffset	Offset into the source data at which to start filling buffer.
lpnBytesWritten	The number of bytes written into the buffer. If it is less than `nBufferSize`, then there is no more data.

Description:

Gets all or part of the image data.

Return Value:

Error count.

IG_PDE_postscript_set_data

Group: PDE PostScript

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_set_data (
HIG_PDE_POSTSCRIPT hPostScript,
LPBYTE lpBuffer,
LONG nBufferSize );
```

Arguments:

hPostScript	PDE postscript object.
lpBuffer	Contains the data.
nBufferSize	Length of the data in bytes.

Description:

Sets the data for `hPostScript`.

Return Value:

Error count.

IG_PDE_postscript_get_data_stream

Group: PDE PostScript

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_get_data_stream (
HIG_PDE_POSTSCRIPT hPostScript,
LPHIG_PDF_STREAM lphStream );
```

Arguments:

hPostScript	PDE postscript object.
lphStream	Stream for hPostScript.

Description:

Gets a stream for the data. The data in the stream is decoded (no filters).

The caller must dispose of the returned stream by calling `IG_PDF_stream_close`.

Return Value:

Error count.

IG_PDE_postscript_set_data_stream

Group: PDE PostScript

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_set_data_stream (
HIG_PDE_POSTSCRIPT hPostScript,
HIG_PDF_STREAM hStream );
```

Arguments:

hPostScript	PDE postscript object.
hStream	Stream for the data.

Description:

Sets a stream for the data. The data must be un-encoded (no filters).

Return Value:

Error count.

HIG_PDE_SOFTMASK

Type: PDE Soft Mask Object

Description:

Handle to the PDE soft mask object. SoftMask– an object for creating and manipulating a soft mask in a PDF file.

Class Members:

IG_PDE_softmask_create	Creates a new soft mask object.
IG_PDE_softmask_create_from_name	Create a new soft mask from a name.
IG_PDE_softmask_get_form	Acquires the form that defines the soft mask.
IG_PDE_softmask_set_form	Sets the form that defines the soft mask.
IG_PDE_softmask_get_backdrop_color	Gets the array of color values of the backdrop color.
IG_PDE_softmask_set_backdrop_color	Sets the backdrop color values.
IG_PDE_softmask_get_name	Gets the soft mask name.

IG_PDE_softmask_create

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_create(
HIG_PDF_DOC hDoc,
LONG nType,
HIG_PDE_FORM hForm,
LPHIG_PDE_SOFTMASK lphSoftMask );
```

Arguments:

hDoc	The container document.
nType	Specifies how the mask is to be computed. One of the `enumIGPDESoftMaskCreateFlags`.
hForm	The form XObject that defines the soft mask. It is the source of the mask values and the color space in which the composite computation is to be done.
lphSoftMask	The newly created object.

Description:

Creates a new soft mask object.

Call `IG_PDE_element_release` to dispose the created object when finished with it.

Return Value:

Error count.

IG_PDE_softmask_create_from_name

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_create_from_name(
HIG_PDF_ATOM hName,
LPHIG_PDE_SOFTMASK lphSoftMask );
```

Arguments:

hName	The new name for the soft mask. **NOTE:** Currently, the only valid name is **None**.
lphSoftMask	The newly created object.

Description:

Create a new soft mask from a name.

Call `IG_PDE_element_release` to dispose the created object when finished with it.

Return Value:

Error count.

IG_PDE_softmask_get_form

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_get_form(
HIG_PDE_SOFTMASK hSoftMask,
LPAT_PDF_FIXEDMATRIX lpMatrix,
LPHIG_PDE_FORM lphForm );
```

Arguments:

hSoftMask	The soft mask object.
lpMatrix	Matrix defining the transformation from coordinate space to user space.
lphForm	The XObject form of the soft mask.

Description:

Acquires the form that defines the soft mask.

Return Value:

Error count.

IG_PDE_softmask_set_form

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_set_form(
HIG_PDE_SOFTMASK hSoftMask,
HIG_PDE_FORM hForm );
```

Arguments:

hSoftMask	The soft mask object.
hForm	The form XObject.

Description:

Sets the form that defines the soft mask.

Return Value:

Error count.

IG_PDE_softmask_get_backdrop_color

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_get_backdrop_color (
HIG_PDE_SOFTMASK hSoftMask,
LPAT_PDF_FIXED lpColorValues,
LONG nColorValuesLen,
LPLONG lpnCount );
```

Arguments:

hSoftMask	The soft mask object.
lpColorValues	Pointer to an array of color values. If NULL , the number of color values is returned in `lpnCount`.
nColorValuesLen	Length of the array `lpColorValues`.
lpnCount	Number of values copied.

Description:

Gets the array of color values of the backdrop color. Given a pointer to an array and the length of the array, copies the color values to that array and returns the number of values copied. If the pointer to the array is NULL, the number of color values is returned in `lpnCount`.

Return Value:

Error count.

IG_PDE_softmask_set_backdrop_color

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_set_backdrop_color (
HIG_PDE_SOFTMASK hSoftMask,
LPAT_PDF_FIXED lpColorValues,
LONG nColorValuesLen );
```

Arguments:

hSoftMask	The soft mask object.
lpColorValues	Pointer to an array of color values.
nColorValuesLen	The number of values pointed to by `lpColorValues`.

Description:

Sets the backdrop color values.

Return Value:

Error count.

IG_PDE_softmask_get_name

Group: PDE Soft Mask

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_get_name (
HIG_PDE_SOFTMASK hSoftMask,
LPHIG_PDF_ATOM lphName );
```

Arguments:

hSoftMask	The soft mask object.
lphName	Soft mask name if it is a name; `IG_PDF_ATOM_NULL` otherwise.

Description:

Gets the soft mask name.

Return Value:

Error count.

HIG_PDE_SHADING

Type: PDE Shading Object

Description:

Handle to the PDE shading object. Shading – an element that represents smooth shading.

Class Members:

IG_PDE_shading_create	Creates a smooth shading object.

IG_PDE_shading_create

Group: PDE Shading

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_shading_create (
HIG_PDF_BASOBJ hDictionary,
LPAT_PDF_FIXEDMATRIX lpMatrix,
LPHIG_PDE_SHADING lphShading );
```

Arguments:

hDictionary	The shading dictionary.
lpMatrix	The location and transformation matrix of the shading object.
lphShading	A smooth shading object.

Description:

Creates a smooth shading object.

Call `IG_PDE_element_release` to dispose the created object when finished with it.

Return Value:

Error count.

HIG_PDE_TEXT

Type: PDE Text Object

Description:

Handle to the PDE text object. Text – an element representing text. It is a container for text as show strings or as individual characters. Each sub-element may have different graphics state properties. However, the same clip applies to all sub-elements of a Text. Also, the char path of a Text can be used to represent a clip.

Class Members:

IG_PDE_text_create	Creates an empty text object.
IG_PDE_text_add	Adds a character or a text run to a PDE Text object.
IG_PDE_text_add_item	Adds a text item to a text element at a given index position.
IG_PDE_text_get_advance	Gets the advance width of a character or a text element.
IG_PDE_text_get_bbox	Gets the bounding box of a character or a text run.
IG_PDE_text_get_font	Gets the font for a text character or element.
IG_PDE_text_get_gstate	Gets the graphics state of a character or a text run.
IG_PDE_text_get_item	Obtains a text item from a text element at a given index position.
IG_PDE_text_get_matrix	Returns the matrix of a character or a text element.
IG_PDE_text_get_byte_count	Gets the number of bytes occupied by the character code or text run.
IG_PDE_text_get_char_count	Gets the number of characters in a text object.
IG_PDE_text_get_runs_count	Gets the number of text runs (show strings) in a text object.
IG_PDE_text_get_quad	Gets the quad bounding the specified text run or character.
IG_PDE_text_get_run_for_char	Gets the index of the text run that contains the nth character in a text object.
IG_PDE_text_get_state	Gets the text state of a character or a text element.
IG_PDE_text_get_stroke_matrix	Gets the stroke matrix of a character or a text run.
IG_PDE_text_get_text	Gets the text for a text run or character.
IG_PDE_text_is_at_point	Tests whether a point is on specified text.
IG_PDE_text_is_at_rect	Tests whether any part of a rectangle is on the specified text.
IG_PDE_text_remove	Removes characters or text runs from a text object.
IG_PDE_text_remove_items	Removes contiguous text items from a text element starting at a given index position.
IG_PDE_text_replace_chars	Replaces characters in a text object.
IG_PDE_text_run_get_char_offset	Gets the character offset of the first character of the specified text run.

IG_PDE_text_run_get_char_count	Gets the number of characters in a text run.
IG_PDE_text_run_set_font	Sets the font of a text run.
IG_PDE_text_run_set_gstate	Sets the graphics state of a text run.
IG_PDE_text_run_set_matrix	Sets the matrix of a text run.
IG_PDE_text_run_set_state	Sets the text state of a text run.
IG_PDE_text_run_set_stroke_matrix	Sets the stroke matrix of a text run.
IG_PDE_text_split_run_at	Splits a text run into two text runs.

IG_PDE_text_create

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_create(
LPHIG_PDE_TEXT lphText );
```

Arguments:

lphText	An empty text object.

Description:

Creates an empty text object.

Call `IG_PDE_element_release` to dispose the created object when finished with it.

Return Value:

Error count.

IG_PDE_text_add

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_add(
HIG_PDE_TEXT hText,
UINT nFlag,
UINT nIndex,
LPBYTE lpText,
```

```
UINT nTextLen,
HIG_PDE_FONT hFont,
LPAT_PDE_GRAPHICSTATE lpGstate,
LPAT_PDE_TEXTSTATE lpTstate,
LPAT_PDF_FIXEDMATRIX lpTextMatrix,
LPAT_PDF_FIXEDMATRIX lpStrokeMatrix );
```

Arguments:

hText	Text object to which a character or text run is added.
nFlag	`enumIGPDETextFlag` flag that specifies what kind of text to add. Must be either: `IG_PDE_TEXT_CHAR` — for a text character `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index after which to add character or text run.
lpText	Pointer to the characters to add. **NOTE:** Passing NULL for text can invalidate the text object but will not raise an error. Callers must not pass NULL for this parameter.
nTextLen	Length of the text, in bytes.
hFont	Font for the element.
lpGstate	Pointer to `AT_PDE_GRAPHICSTATE` structure with the graphics state for the element.
lpTstate	Pointer to `AT_PDE_TEXTSTATE` structure with text state for the element.
lpTextMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the matrix for the element.
lpStrokeMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the matrix for the line width when stroking text. May be NULL.

Description:

Adds a character or a text run to a PDE Text object.

Return Value:

Error count.

IG_PDE_text_add_item

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_add_item(
```

```
HIG_PDE_TEXT hText,
UINT nIndex,
HIG_PDE_TEXTITEM hTextItem );
```

Arguments:

hText	Text object to which an item is added.
nIndex	Index of the text item in `hText`.
hTextItem	The text item to add.

Description:

Adds a text item to a text element at a given index position.

Return Value:

Error count.

IG_PDE_text_get_advance

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_advance(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDPOINT lpAdvanceWidth );
```

Arguments:

hText	Text object containing a character or text run whose advance width is found.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run In addition, set the `IG_PDE_TEXT_PAGE_SPACE` flag to obtain the advance width in user space. If it is not set, the advance width is in character space. If this flag is not set, this method returns a value that is independent of any sizes, matrices, or scaling, simply adding up the font's raw glyph widths, supplemented only by unscaled character and word spacing.
nIndex	Index of the character or text run in `hText`.
lpAdvanceWidth	Pointer to `AT_PDF_FIXEDPOINT` value indicating the advance width.

Description:

Gets the advance width of a character or a text element. Advance width is returned in either character space or user space. The advance width is the amount by which the current point advances when the character is drawn.

Advance width may be horizontal or vertical, depending on the writing style. Thus `lpAdvanceWidth` has both a horizontal and vertical component.

Return Value:

Error count.

IG_PDE_text_get_bbox

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_bbox(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDRECT lpBBox );
```

Arguments:

hText	Text object containing a character or text run whose bounding box is found.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpBBox	Pointer to `AT_PDF_FIXEDRECT` to set to the bounding box of specified character or text run.

Description:

Gets the bounding box of a character or a text run.

Return Value:

Error count.

IG_PDE_text_get_font

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_font(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPHIG_PDE_FONT lphFont);
```

Arguments:

hText	Text object containing a character or text run whose font is found.
nFlags	Specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ IG_PDE_TEXT_CHAR — for a text character ■ IG_PDE_TEXT_RUN — for a text run
nIndex	Index of the character or text run in the text object.
lphFont	HIG_PDE_FONT return value.

Description:

Gets the font for a text character or element.

Return Value:

Error count.

IG_PDE_text_get_gstate

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_gstate(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDE_GRAPHICSTATE lpGstate );
```

Arguments:

hText	Text object containing a character or text run whose graphics state is obtained.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpGstate	Pointer to a `AT_PDE_GRAPHICSTATE` structure with graphics state of specified character or text run.

Description:

Gets the graphics state of a character or a text run.

NOTE: Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this method. Be sure to release these non-NULL objects when disposing of lpGstate.

Return Value:

Error count.

IG_PDE_text_get_item

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_item(
HIG_PDE_TEXT hText,
UINT nIndex,
LPHIG_PDE_TEXTITEM lphTextItem );
```

Arguments:

hText	Text object from which the text item is obtained.
nIndex	Index of the text item in `hText`.
lphTextItem	The text item object.

Description:

Obtains a text item from a text element at a given index position.

Return Value:

Error count.

IG_PDE_text_get_matrix

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_matrix(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hText	Text object containing a character or text run whose matrix is obtained.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the matrix of specified character or text run.

Description:

Returns the matrix of a character or a text element.

Return Value:

Error count.

IG_PDE_text_get_byte_count

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_byte_count(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPUINT lpnByteCount );
```

Arguments:

hText	Text object containing a character or text run whose text is examined.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpnByteCount	Number of bytes occupied by the text run or character.

Description:

Gets the number of bytes occupied by the character code or text run.

Return Value:

Error count.

IG_PDE_text_get_char_count

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_char_count(
HIG_PDE_TEXT hText,
LPUINT lpnCharCount );
```

Arguments:

hText	Text object containing a character or text run whose text is examined.
lpnCharCount	Total number of characters in `hText`.

Description:

Gets the number of characters in a text object.

Return Value:

Error count.

IG_PDE_text_get_runs_count

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_runs_count(
HIG_PDE_TEXT hText,
LPUINT lpnRunsCount);
```

Arguments:

hText	Text object whose number of text runs is found.
lpnRunsCount	Number of text runs in hText.

Description:

Gets the number of text runs (show strings) in a text object.

Return Value:

Error count.

IG_PDE_text_get_quad

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_quad(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDQUAD lpQuad );
```

Arguments:

hText	Text object containing a character or text run whose text is examined.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run In addition, if the `IG_PDE_TEXT_GET_BOUNDS` flag is set, this method uses the font descriptor's FontBBox, which is the smallest rectangle that encloses all characters in the font. The advance portion is based on the x-coordinates of the left and right sides of FontBBox and the advance width.
nIndex	Index of the character or text run in the text object.
lpQuad	Pointer to `AT_PDF_FIXEDQUAD` that bounds the specified character or text run.

Description:

Gets the quad bounding the specified text run or character.

The advance portion of the quad is based on the left side bearing and advance width.

Return Value:

Error count.

IG_PDE_text_get_run_for_char

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_run_for_char(
HIG_PDE_TEXT hText,
UINT nCharIndex,
LPUINT lpnRunIndex );
```

Arguments:

hText	Text object containing a character or text run whose text is examined.
nCharIndex	Number of the character to find in `hText`.
lpnRunIndex	Index of the text run with the specified character index in `hText`.

Description:

Gets the index of the text run that contains the nth character in a text object.

Return Value:

Error count.

IG_PDE_text_get_state

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_state(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDE_TEXTSTATE lpTstate );
```

Arguments:

hText	Text object containing a character or text run whose text is examined.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpTstate	Pointer to `AT_PDE_TEXTSTATE` structure to fill with the text state of the specified character or text run.

Description:

Gets the text state of a character or a text element.

Return Value:

Error count.

IG_PDE_text_get_stroke_matrix

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_stroke_matrix(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hText	Text object containing a character or text run whose text is examined.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpMatrix	Pointer to `AT_PDF_FIXEDMATRIX` that holds the stroke matrix of specified character or text run. This matrix is the transformation for line widths when stroking. The h and v values of the matrix are ignored.

Description:

Gets the stroke matrix of a character or a text run.

Return Value:

Error count.

IG_PDE_text_get_text

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_text(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPBYTE lpText,
LPUINT lpnTextLen );
```

Arguments:

hText	Text object containing a character or text run whose text is found.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpText	Text of specified character or text run. `lpText` must be large enough to hold the returned text.
lpnTextLen	Number of bytes in text run or character.

Description:

Gets the text for a text run or character.

Return Value:

Error count.

IG_PDE_text_is_at_point

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_is_at_point(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDPOINT lpPoint,
LPAT_BOOL lpbResult );
```

Arguments:

hText	Text object containing a character or text run.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpPoint	The point, specified in user space coordinates.
lpbResult	True if the point is on the text, False otherwise.

Description:

Tests whether a point is on specified text. Checks if the point is in a bounding box for `hText`.

Return Value:

Error count.

IG_PDE_text_is_at_rect

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_is_at_rect(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPAT_PDF_FIXEDRECT lpFixedRect,
LPAT_BOOL lpbResult );
```

Arguments:

hText	Text object containing a character or text run.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpFixedRect	The rectangle, specified in user space coordinates.
lpbResult	True if the text is on the rectangle, False otherwise.

Description:

Tests whether any part of a rectangle is on the specified text.

Return Value:

Error count.

IG_PDE_text_remove

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_remove(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
UINT nCount );
```

Arguments:

hText	Text object containing a character or text run.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
nCount	Number of characters or text runs to remove.\

Description:

Removes characters or text runs from a text object.

Return Value:

Error count.

IG_PDE_text_remove_items

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_remove_items(
HIG_PDE_TEXT hText,
UINT nIndex,
UINT nCount );
```

Arguments:

hText	Text object from which the text items are removed.
nIndex	Index of the first text item in pdeText to remove.
nCount	The number of text items to remove.

Description:

Removes contiguous text items from a text element starting at a given index position.

Return Value:

Error count.

IG_PDE_text_replace_chars

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_replace_chars(
HIG_PDE_TEXT hText,
UINT nFlags,
UINT nIndex,
LPBYTE lpText,
UINT nTextLen );
```

Arguments:

hText	Text object containing a character or text run.
nFlags	`enumIGPDETextFlag` value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ■ `IG_PDE_TEXT_CHAR` — for a text character ■ `IG_PDE_TEXT_RUN` — for a text run
nIndex	Index of the character or text run in the text object.
lpText	Replacement text.
nTextLen	Number of bytes to replace.

Description:

Replaces characters in a text object.

This method does not change the number of characters in the text object—extra characters are ignored.

Return Value:

Error count.

IG_PDE_text_run_get_char_offset

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_get_char_offset(
HIG_PDE_TEXT hText,
UINT nRunIndex,
LPUINT lpnOffset );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
lpnOffset	Character offset of the first character of the specified text run in `hText`.

Description:

Gets the character offset of the first character of the specified text run.

Return Value:

Error count.

IG_PDE_text_run_get_char_count

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_get_char_count(
HIG_PDE_TEXT hText,
UINT nRunIndex,
LPUINT lpnCount );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
lpnCount	Number of characters in the specified text run.

Description:

Gets the number of characters in a text run.

Return Value:

Error count.

IG_PDE_text_run_set_font

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_font(
```

```
HIG_PDE_TEXT hText,
UINT nRunIndex,
HIG_PDE_FONT hFont );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
hFont	Font set for the text run.

Description:

Sets the font of a text run.

Return Value:

Error count.

IG_PDE_text_run_set_gstate

Group: PDE Text

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_gstate(
HIG_PDE_TEXT hText,
UINT nRunIndex,
LPAT_PDE_GRAPHICSTATE lpGstate );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
lpGstate	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state to set.

Description:

Sets the graphics state of a text run.

Return Value:

Error count.

IG_PDE_text_run_set_matrix

Group: IG PDE

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_matrix(
HIG_PDE_TEXT hText,
UINT nRunIndex,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
lpMatrix	Pointer to AT_PDF_FIXEDMATRIX structure with matrix to set.

Description:

Sets the matrix of a text run.

Return Value:

Error count.

IG_PDE_text_run_set_state

Group: IG PDE

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_state(
HIG_PDE_TEXT hText,
UINT nRunIndex,
LPAT_PDE_TEXTSTATE lpState );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
lpState	Pointer to AT_PDE_TEXTSTATE structure with state to set.

Description:

Sets the text state of a text run.

Return Value:

Error count.

IG_PDE_text_run_set_stroke_matrix

Group: IG PDE

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_stroke_matrix(
HIG_PDE_TEXT hText,
UINT nRunIndex,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hText	Text object containing a text run.
nRunIndex	Index of the text run.
lpMatrix	Pointer to `AT_PDF_FIXEDMATRIX` structure with store matrix to set.

Description:

Sets the stroke matrix of a text run.

Return Value:

Error count.

IG_PDE_text_split_run_at

Group: IG PDE

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_split_run_at(
HIG_PDE_TEXT hText,
UINT nSplitLoc );
```

Arguments:

hText	Text object containing a text run.
nSplitLoc	Split location, relative to the text object. The first text run is from character index 0 up to `nSplitLoc`. The second text run is from `nSplitLoc` + 1 to the end of the run.

Description:

Splits a text run into two text runs.

Return Value:

Error count.

HIG_PDE_TEXTITEM

Type: PDE Text Item Object

Description:

Handle to the PDE text item object. TextItem – a PDE element representing a text object.

Class Members:

IG_PDE_textitem_create	Creates a text item element containing a character or text run, which can be added to a PDE Text object.
IG_PDE_textitem_copy_text	Copies the text from a text item element into a character buffer.
IG_PDE_textitem_get_font	Gets the font for a text item.
IG_PDE_textitem_set_font	Sets a font for a text item.
IG_PDE_textitem_get_gstate	Gets the graphics state for a text item.
IG_PDE_textitem_set_gstate	Sets the graphics state of a text item.
IG_PDE_textitem_get_text_length	Gets the text length for a text item.
IG_PDE_textitem_get_matrix	Gets the text matrix for a character in a text item.
IG_PDE_textitem_set_matrix	Sets the text matrix for a text item.
IG_PDE_textitem_get_state	Gets the text state of a text item.
IG_PDE_textitem_set_state	Sets the text state of a text item.
IG_PDE_textitem_remove_chars	Removes contiguous characters from a text item.
IG_PDE_textitem_replace_chars	Replaces characters in a text item.
IG_PDE_textitem_replace_text	Replaces all of the text in a text item.

IG_PDE_textitem_create

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_create(
```

```
LPBYTE lpText,
UINT nTextLen,
HIG_PDE_FONT hFont,
LPAT_PDE_GRAPHICSTATE lpGstate,
LPAT_PDE_TEXTSTATE lpTstate,
LPAT_PDF_FIXEDMATRIX lpTextMatrix
LPHIG_PDE_TEXTITEM lphTextItem );
```

Arguments:

lpText	Pointer to the characters to add. **NOTE:** Passing NULL for text can invalidate the text object but will not raise an error. Callers must not pass NULL for this parameter.
nTextLen	Length of the text, in bytes.
hFont	Font for the element.
lpGstate	Pointer to AT_PDE_GRAPHICSTATE structure with the graphics state for the element.
lpTstate	Pointer to AT_PDE_TEXTSTATE structure with text state for the element.
lpTextMatrix	Pointer to AT_PDF_FIXEDMATRIX that holds the matrix for the element.
lphTextItem	A text element object.

Description:

Creates a text item element containing a character or text run, which can be added to a PDE Text object.

Call **IG_PDE_element_release** to dispose the created object when finished with it.

Return Value:

Error count.

IG_PDE_textitem_copy_text

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_copy_text(
HIG_PDE_TEXTITEM hTextItem,
LPBYTE lpText,
UINT nTextLen,
LPUINT lpnTextItemLen );
```

Arguments:

hTextItem	Text item object.
lpText	A pointer to a buffer in which to store the copy.
nTextLen	Length of the text buffer, in bytes.
lpnTextItemLen	The length in bytes of hTextItem.

Description:

Copies the text from a text item element into a character buffer.

Return Value:

Error count.

IG_PDE_textitem_get_font

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_font(
HIG_PDE_TEXT hTextItem,
LPHIG_PDE_FONT lphFont);
```

Arguments:

hTextItem	Text item whose font is obtained.
lphFont	Font for hTextItem.

Description:

Gets the font for a text item.

Return Value:

Error count.

IG_PDE_textitem_set_font

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_font(
HIG_PDE_TEXT hTextItem,
HIG_PDE_FONT hFont );
```

Arguments:

hTextItem	Text object containing a text run.
hFont	Font set for a text item.

Description:

Sets a font for a text item.

Return Value:

Error count.

IG_PDE_textitem_get_gstate

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_gstate(
HIG_PDE_TEXT hTextItem,
LPAT_PDE_GRAPHICSTATE lpGstate );
```

Arguments:

hTextItem	Text item whose graphic state is obtained.
lpGstate	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state of the text item.

Description:

Gets the graphics state for a text item.

NOTE: Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this method. Be sure to release these non-NULL objects when disposing of lpGstate.

Return Value:

Error count.

IG_PDE_textitem_set_gstate

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_gstate(
HIG_PDE_TEXTITEM hTextItem,
LPAT_PDE_GRAPHICSTATE lpGstate );
```

Arguments:

hTextItem	Text item whose graphic state is set.
lpGstate	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state to set.

Description:

Sets the graphics state of a text item.

Return Value:

Error count.

IG_PDE_textitem_get_text_length

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_text_length(
HIG_PDE_TEXT hTextItem,
LPLONG lpnLength );
```

Arguments:

hTextItem	Text item whose length is obtained.
lpnLength	The text length, in bytes.

Description:

Gets the text length for a text item.

Return Value:

Error count.

IG_PDE_textitem_get_matrix

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_matrix(
HIG_PDE_TEXTITEM hTextItem,
UINT nCharOffset,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hTextItem	The text item.
nCharOffset	The offset of the character whose text matrix is obtained.
lpMatrix	Pointer to AT_PDF_FIXEDMATRIX with text matrix of the character.

Description:

Gets the text matrix for a character in a text item.

Return Value:

Error count.

IG_PDE_textitem_set_matrix

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_matrix(
HIG_PDE_TEXTITEM hTextItem,
LPAT_PDF_FIXEDMATRIX lpMatrix );
```

Arguments:

hTextItem	The text item.
lpMatrix	Pointer to AT_PDF_FIXEDMATRIX with the new text matrix of the text item.

Description:

Sets the text matrix for a text item.

Return Value:

Error count.

IG_PDE_textitem_get_state

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_state(
HIG_PDE_TEXTITEM hTextItem,
LPAT_PDE_TEXTSTATE lpTstate );
```

Arguments:

hTextItem	The text item.
lpTstate	Pointer to AT_PDE_TEXTSTATE structure with text state of the text item..

Description:

Gets the text state of a text item.

Return Value:

Error count.

IG_PDE_textitem_set_state

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_state(
HIG_PDE_TEXTITEM hTextItem,
LPAT_PDE_TEXTSTATE lpTstate );
```

Arguments:

hTextItem	The text item.
lpTstate	Pointer to AT_PDE_TEXTSTATE structure with new text state of the text item.

Description:

Sets the text state of a text item.

Return Value:

Error count.

IG_PDE_textitem_remove_chars

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_remove_chars (
HIG_PDE_TEXTITEM hTextItem,
UINT nCharOffset,
UINT nCount );
```

Arguments:

hTextItem	The text item.
nCharOffset	Offset of the first character to remove.
nCount	The number of characters to remove.

Description:

Removes contiguous characters from a text item.

Return Value:

Error count.

IG_PDE_textitem_replace_chars

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_replace_chars (
HIG_PDE_TEXTITEM hTextItem,
UINT nCharIndex,
LPBYTE lpNewChars,
UINT nNewCharsLen);
```

Arguments:

hTextItem	The text item.
nCharIndex	Index position of the characters to replace.
lpNewChars	Replacement text.
nNewCharsLen	Number of bytes to replace.

Description:

Replaces characters in a text item.

This method does not change the number of characters in the text item—extra characters are ignored.

Return Value:

Error count.

IG_PDE_textitem_replace_text

Group: PDE Text Item

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_replace_text (
HIG_PDE_TEXTITEM hTextItem,
LPBYTE lpNewText,
UINT nNewTextLen);
```

Arguments:

hTextItem	The text item.
lpNewText	Replacement text.
nNewTextLen	Number of bytes to replace.

Description:

Replaces all of the text in a text item.

Return Value:

Error count.

HIG_PDE_XGROUP

Type: PDE XGroup Object

Description:

Handle to the PDE XGroup object. XGroup – a transparency (XGroup) resource.

Class Members:

IG_PDE_xgroup_create	Creates a new XGroup of the given type.
IG_PDE_xgroup_get_colorspace	Acquires the color space of the transparency group.
IG_PDE_xgroup_set_colorspace	Sets the color space for the XGroup.
IG_PDE_xgroup_get_isolated	Gets the isolated boolean value of the transparency group.
IG_PDE_xgroup_set_isolated	Sets the XGroup to be isolated or not.
IG_PDE_xgroup_get_knockout	Gets the knockout boolean value of the transparency group.
IG_PDE_xgroup_set_knockout	Sets the knockout value.

IG_PDE_xgroup_create

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_create(
HIG_PDF_DOC hDoc,
LONG nType,
LPHIG_PDE_XGROUP lphXGroup );
```

Arguments:

hDoc	The document the object will be created in.
nType	enumIGPDEXGroupCreateFlags value.
lphXGroup	The newly created XGroup object.

Description:

Creates a new XGroup of the given type.

Call **IG_PDE_element_release** to dispose the created object when finished with it.

Return Value:

Error count.

IG_PDE_xgroup_get_colorspace

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_get_colorspace (
HIG_PDE_XGROUP hXGroup,
LPHIG_PDE_COLORSPACE lphColorSpace );
```

Arguments:

hXGroup	The transparency group object
lphColorSpace	The color space.

Description:

Acquires the color space of the transparency group.

Return Value:

Error count.

IG_PDE_xgroup_set_colorspace

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_set_colorspace (
HIG_PDE_XGROUP hXGroup,
HIG_PDE_COLORSPACE hColorSpace );
```

Arguments:

hXGroup	The transparency group object
hColorSpace	The color space to associate with the XGroup.

Description:

Sets the color space for the XGroup.

Return Value:

Error count.

IG_PDE_xgroup_get_isolated

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_get_isolated (
HIG_PDE_XGROUP hXGroup,
LPAT_PDF_BOOL lpbIsolated );
```

Arguments:

hXGroup	The transparency group object
lpbIsolated	True if the transparency group is isolated; False otherwise.

Description:

Gets the isolated boolean value of the transparency group.

Return Value:

Error count.

IG_PDE_xgroup_set_isolated

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_set_isolated (
HIG_PDE_XGROUP hXGroup,
AT_PDF_BOOL bIsolated );
```

Arguments:

hXGroup	The transparency group object
bIsolated	True to isolate the XGroup, False otherwise.

Description:

Sets the XGroup to be isolated or not. Corresponds to the /I key within the XGroup's dictionary.

Return Value:

Error count.

IG_PDE_xgroup_get_knockout

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_get_knockout(
HIG_PDE_XGROUP hXGroup,
LPAT_PDF_BOOL lpbKnockout );
```

Arguments:

hXGroup	The transparency group object
lpbKnockout	The knockout value

Description:

Gets the knockout boolean value of the transparency group.

Return Value:

Error count.

IG_PDE_xgroup_set_knockout

Group: PDE XGroup

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_set_knockout(
HIG_PDE_XGROUP hXGroup,
AT_PDF_BOOL bKnockout);
```

Arguments:

hXGroup	The transparency group object
bKnockout	The knockout value.

Description:

Sets the knockout value.

Return Value:

Error count.

HIG_PDE_XOBJECT

Type: PDE XObject Object

Description:

Handle to the PDE XObject object. XObject – an element representing an arbitrary XObject.

Class Members:

IG_PDE_xobject_create	Creates a new XObject from hObject.

IG_PDE_xobject_create

Group: PDE XObject

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xobject_create(
HIG_PDF_BASOBJ hBasObj,
LPHIG_PDE_XOBJECT lphXObject );
```

Arguments:

hBasObj	Base object for XObject.
lphXObject	XObject corresponding to the hBasObj.

Description:

Creates a new XObject from hObject.

Call **IG_PDE_element_release** to dispose the created object when finished with it.

Return Value:

Error count.

Subsidiary Types / DLL Structures

Simple structure types that are used for creating, attributing and manipulating general and editing objects.

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

Subsidiary Types / DLL Structures	
AT_PDF_BOOL	Boolean type with two values: TRUE (1) or FALSE (0)
AT_PDF_FIXED	The Fixed type is a 32-bit quantity representing a rational number with the high (low on little-endian machines) 16 bits representing the number's mantissa and the low (high) 16 bits representing the fractional part. The definition is platform-dependent. LPAT_PDF_FIXED is a pointer to an AT_PDF_FIXED. Addition, subtraction, and negation with AT_PDF_FIXED types can be done with + and -, unless you care about overflow. (Overflow in Fixed-value operations is indicated by the values IG_PDF_FIXED_POSITIVE_INFINITY and IG_PDF_FIXED_NEGATIVE_INFINITY.)
AT_PDF_FIXEDPOINT	Point (in two-dimensional space) represented by two fixed numbers.
AT_PDF_FIXEDMATRIX	Matrix containing fixed numbers.
AT_PDF_FIXEDRECT	A rectangle represented by the coordinates of its four sides. A rectangle differs from a quadrilateral in that the former must always have horizontal and vertical sides, and opposite sides must be parallel.
AT_PDF_FIXEDQUAD	Quadrilateral represented by four fixed points (one at each corner). A quadrilateral differs from a rectangle in that the latter must always have horizontal and vertical sides, and opposite sides must be parallel.
AT_PDF_FONT_STYLES	Font styles.
AT_PDF_FONT_METRICS	Font metrics.
AT_PDF_PRINTOPTIONS	This structure used to provide printing parameters for IG_PDF_doc_print function.
AT_PDF_PRINTPARAMS	This structure indicates how a document should be printed.
AT_PDF_SECURITYDATA	Describes the data for the standard security handler.
AT_PDE_FONTATTRS	Attributes of a PDE Font and a PDF SysFont.
AT_PDE_COLORVALUE	Color value.
AT_PDE_COLORSPEC	Color specification.

AT_PDE_DASH	Dash specification, as described in Table 4.8 in the PDF Reference. See Section 4.3.2 for more information on line dash patterns.
AT_PDE_GRAPHICSTATE	Attributes of a PDE Element or a PDE Text sub-element.
AT_PDE_TEXTSTATE	Attributes of a PDE Text element.
AT_PDF_SYSFONT_PLATDATA	PDF SysFont platform specific data.
AT_PDE_FONT_INFO	PDE Font information.
AT_PDE_FONT_CREATEFROMSYSFONTPARAMS	Parameters for PDE font creation.
AT_PDE_COLORRANGE	Color range.
AT_PDE_XYZCOLOR	XYZ color.
AT_PDE_COLORDATA_CALGRAY	CalGray color space.
AT_PDE_COLORDATA_CALRGB	CalRGB color space.
AT_PDE_COLORDATA_LAB	L*a*b* color space.
AT_PDE_COLORDATA_ICCBASED	ICC based color space.
AT_PDE_COLORDATA_INDEXED	Indexed color space.
AT_PDE_COLORDATA_SEPARATION	Separation color space
AT_PDE_COLORDATA_DEVICEN	DeviceN color space.
AT_PDE_COLORDATA	Fill in one of the following members of this union, then pass it to PDE ColorSpace creation routine. Please see section 7.10 of the PDF Reference Manual for information on color spaces.
AT_PDE_PSATTRS	Attributes of a PDE PS object.
AT_PDE_IMAGEATTRS	Attributes of a PDE Image object.
AT_PDE_FILTERSPEC	Filter element in a filter array.
AT_PDE_FILTERARRAY	Filter information for streams. Array of FilterSpec elements. Usually consists of 2 filter elements: text encoding and image compression.
AT_PDE_CONTENTATTRS	Attributes of a PDE Content object.

AT_PDF_FIXEDPOINT

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_FIXEDPOINT
{
   AT_PDF_FIXED h;
```

```
   AT_PDF_FIXED v;
} AT_PDF_FIXEDPOINT;
typedef AT_PDF_FIXEDPOINT FAR* LPAT_PDF_FIXEDPOINT;
```

Arguments:

AT_PDF_FIXED	h	Horizontal value
AT_PDF_FIXED	v	Vertical value

Description:

Point (in two-dimensional space) represented by two fixed numbers.

AT_PDF_FIXEDMATRIX

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_FIXEDMATRIX
{
   AT_PDF_FIXED a;
   AT_PDF_FIXED b;
   AT_PDF_FIXED c;
   AT_PDF_FIXED d;
   AT_PDF_FIXED h;
   AT_PDF_FIXED v;
} AT_PDF_FIXEDMATRIX;
typedef AT_PDF_FIXEDMATRIX FAR* LPAT_PDF_FIXEDMATRIX;
```

Arguments:

AT_PDF_FIXED	a	A value
AT_PDF_FIXED	b	B value
AT_PDF_FIXED	c	C value
AT_PDF_FIXED	d	D value
AT_PDF_FIXED	h	H value
AT_PDF_FIXED	v	V value

Description:

Matrix containing fixed numbers.

AT_PDF_FIXEDRECT

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_FIXEDRECT
{
   AT_PDF_FIXED left;
   AT_PDF_FIXED top;
   AT_PDF_FIXED right;
   AT_PDF_FIXED bottom;
} AT_PDF_FIXEDRECT;
typedef AT_PDF_FIXEDRECT FAR* LPAT_PDF_FIXEDRECT;
```

Arguments:

AT_PDF_FIXED	left	Left value
AT_PDF_FIXED	right	Right value
AT_PDF_FIXED	top	Top value
AT_PDF_FIXED	bottom	Bottom value

Description:

A rectangle represented by the coordinates of its four sides. A rectangle differs from a quadrilateral in that the former must always have horizontal and vertical sides, and opposite sides must be parallel.

AT_PDF_FIXEDQUAD

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_FIXEDQUAD
{
   AT_PDF_FIXEDPOINT tl, tr, bl, br;
} AT_PDF_FIXEDQUAD;
typedef AT_PDF_FIXEDQUAD FAR* LPAT_PDF_FIXEDQUAD;
```

Arguments:

AT_PDF_FIXEDPOINT	tl	Top left value
AT_PDF_FIXEDPOINT	tr	Top right value
AT_PDF_FIXEDPOINT	bl	Bottom left value
AT_PDF_FIXEDPOINT	br	Bottom right value

Description:

Quadrilateral represented by four fixed points (one at each corner). A quadrilateral differs from a rectangle in that the latter must always have horizontal and vertical sides, and opposite sides must be parallel.

AT_PDF_FONT_STYLES

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_FONT_STYLES
{
   AT_BYTE sFamilyClassID;
   AT_BYTE sFamilySubclassID;
   AT_BYTE bFamilyType;
   AT_BYTE bSerifStyle;
   AT_BYTE bWeight;
   AT_BYTE bProportion;
} AT_PDF_FONT_STYLES;
```

Arguments:

sFamilyClassID	Number that identifies the font family and determines the meaning of the remaining Panose digits. Possible families are Latin, Kanji, Hebrew, and so forth.
sFamilySubclassID	Number to identify the kind of family: text, decorative, handwritten, symbols, and so on.
bFamilyType	Number to identify the family type: text, decorative, handwritten, symbols, and so on.
bSerifStyle	Number that specifies the font's serif style, such as cove, obtuse cove, square, bone, and so forth.
bWeight	Number that specifies the font's weight, such as very light, heavy, black, and so on.
bProportion	Number that specifies the font's proportions, such as modern, expanded, condensed, mono spaced, and so on.

AT_PDF_FONT_METRICS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_FONT_METRICS
{
   UINT         flags;
   AT_PDF_FIXEDRECTfontBBox;
   AT_INT16  missingWidth;
   AT_INT16  stemV;
   AT_INT16  stemH;
   AT_INT16  capHeight;
   AT_INT16  xHeight;
   AT_INT16  ascent;
   AT_INT16  descent;
   AT_INT16  leading;
   AT_INT16  maxWidth;
   AT_INT16  avgWidth;
   AT_INT16  italicAngle;
   AT_PDF_FONT_STYLESstyle;
   AT_INT16  baseLineAdj;
} AT_PDF_FONT_METRICS;
```

Arguments:

flags	Must be an OR of the Font Flags values. All unused flags must be off.
fontBBox	Font bounding box in 1000 EM units. (An EM is a typographic unit of measurement equal to the size of a font. In a 12-point font, an EM is 12 points.)
missingWidth	Width of missing character.
stemV	Vertical stem width.
stemH	Horizontal stem width.
capHeight	Capital height.
xHeight	X height.
ascent	Max ascender height.
descent	Max descender depth.
Leading	Additional leading between lines.
maxWidth	Maximum character width.
avgWidth	Average character width.
italicAngle	Italic angle in degrees, if any.
style	Panose and sFamily class values.
baseLineAdj	Baseline adjustment, which is a vertical adjustment for font baseline difference and writing mode 1 (vertical). This should only be used for CIDFontType 2 fonts with font substitution.

AT_PDF_PRINTOPTIONS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_PRINTOPTIONS
{
   AT_DWORD size;
   AT_PDF_BOOL emitToFile;
   LPAT_PDF_PRINTPARAMS printParams;
   HIG_PDF_STREAM printStm;
   AT_WORD paperWidth;
   AT_WORD paperHeight;
   AT_DWORD dontEmitListLen;
   char ** dontEmitList;
   AT_PDF_BOOL emitToPrinter;
   char *inFileName;
   char *outFileName;
```

```
    char *deviceName;
    char *driverName;
    char *portName;
    DEVMODE *pDevMode;
    int   startPage;
    int   endPage;
    int   shrinkToFit;
    int   printAnnots;
    int   psLevel;
    int   nCopies;
    int   binaryOK;
    int   emitHalftones;
    int   reverse;
    BYTE farEastFontOpt;
    int   transQuality;
    int   doOPP;
    LPVOID cancelProc;
    LPVOID clientData;
    int   startResult;
    AT_PDF_BOOL forceGDIPrint;
    HDC hDC;
} AT_PDF_PRINTOPTIONS;
typedef AT_PDF_PRINTOPTIONS FAR* LPAT_PDF_PRINTOPTIONS;
```

Arguments:

Description:

This structure used to provide printing parameters for IG_PDF_doc_print function.

To print, set printing method to printer, and fill in inFileName, outFileName, deviceName, driverName, portName, dpi, nCopies, startPage, endPage, psLevel, shrinkToFit, binaryOK, emitHalftones.

AT_PDF_PRINTPARAMS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_PRINTPARAMS
{
    AT_UINT size;
    AT_PDF_PAGE_RANGE *ranges;
    AT_INT32 numRanges;
    AT_PDF_BOOL shrinkToFit;
    AT_PDF_BOOL expandToFit;
    AT_PDF_BOOL rotateAndCenter;
    AT_INT16 printWhat;
    AT_INT16 printWhatAnnot;
    AT_PDF_BOOL emitPS;
```

```
AT_INT32 psLevel;
AT_INT16 outputType;
AT_INT16 incBaseFonts;
AT_INT16 incEmbeddedFonts;
AT_INT16 incType1Fonts;
AT_INT16 incType3Fonts;
AT_INT16 incTrueTypeFonts;
AT_INT16 incCIDFonts;
AT_INT16 incProcsets;
AT_INT16 incOtherResources;
AT_INT32 fontPerDocVM;
AT_PDF_BOOL emitShowpage;
AT_PDF_BOOL emitTTFontsFirst;
AT_PDF_BOOL setPageSize;
AT_PDF_BOOL emitDSC;
AT_PDF_BOOL setupProcsets;
AT_PDF_BOOL emitColorSeps;
AT_PDF_BOOL binaryOK;
AT_PDF_BOOL useSubFileDecode;
AT_PDF_BOOL emitRawData;
AT_PDF_BOOL TTasT42;
AT_FLOAT scale;
AT_PDF_BOOL emitExternalStreamRef;
AT_PDF_BOOL emitHalftones;
AT_PDF_BOOL emitPSXObjects;
AT_PDF_BOOL centerCropBox;
AT_PDF_BOOL emitSeparableImagesOnly;
AT_PDF_BOOL emitDeviceExtGState;
AT_PDF_FIXEDRECT boundingBox;
AT_PDF_BOOL useFontAliasNames;
AT_PDF_BOOL emitPageRotation;
AT_PDF_BOOL reverse;
AT_PDF_FIXEDRECT *tCropBox;
AT_PDF_BOOL emitPageClip;
AT_PDF_BOOL emitTransfer;
AT_PDF_BOOL emitBG;
AT_PDF_BOOL emitUCR;
AT_INT16 farEastFontOpt;
AT_PDF_BOOL suppressCJKSubstitution;
AT_PDF_BOOL suppressCSA;
AT_PDF_BOOL hostBased;
HIG_PDF_ATOM hostBasedOutputCS;
AT_INT16 duplex;
AT_INT16 doTiling;
LPAT_PDF_TILEEX tileInfo;
AT_PDF_BOOL rotate;
AT_PDF_BOOL hostBasedCM;
char  destProfile[256];
HIG_PDF_ATOM destCSAtom;
AT_PDF_BOOL saveVM;
AT_PDF_BOOL doOPP;
AT_INT32 suppressOPPWhenNoSpots;
```

```
    AT_PDF_BOOL optimizeForSpeed;
    AT_PDF_BOOL brokenCRDs;
    AT_PDF_BOOL useMaxVM;
    AT_INT32 lastWidth;
    AT_INT32 lastHeight;
    AT_DWORD bitmapResolution;
    AT_DWORD gradientResolution;
    AT_DWORD transparencyQuality;
    AT_DWORD ocContext;
    AT_PDF_BOOL applyOCGPrintOverrides;
    AT_PDF_BOOL useFullResolutionJP2KData;
    AT_PDF_BOOL emitInRipSeps;
    AT_DWORD whichMarks;
    AT_PDF_BOOL westernMarksStyle;
    AT_PDF_BOOL doProofing;
    char proofProfile[256];
    AT_PDF_BOOL inkBlack;
    AT_PDF_BOOL paperWhite;
    AT_PDF_BOOL useExecForm;
    LPAT_PDF_FLATTEN flattenInfo;
    AT_PDF_BOOL negative;
    AT_INT16 mirrorprint;
    AT_DWORD numCollatedCopies;
    AT_PDF_BOOL emitFlatness;
    AT_INT32 trapType;
    AT_PDF_BOOL TTasCIDT2;
    AT_DWORD markStyle;
    AT_FLOAT lineWidth;
    AT_PDF_BOOL macQDPrinter;

} AT_PDF_PRINTPARAMS;
typedef AT_PDF_PRINTPARAMS FAR* LPAT_PDF_PRINTPARAMS;
```

Arguments:

size	Size of the data structure. Must be set to sizeof(AT_PDF_PRINTPARAMS).
ranges	Ranges of pages to print. Use NULL to print the entire document.
numRanges	Number of ranges of pages to print in ranges. The default value is 0.
shrinkToFit	True if the page is scaled to fit the printer page size, false otherwise. This field overrides scale. The default value is false.
expandToFit	True if small pages are to be scaled up to fit the printer page size, false otherwise. Overrides scale. The default value is false.
rotateAndCenter	True if page is to be rotated to fit printer's orientation, and centered in printer's page size; false otherwise.The default value is false. Rotation and centering (true) only occur, however, if the page contents are too wide to fit on a narrow page (or vice versa) and the page contents are less than an inch smaller than the target page in one direction.
printWhat	enumIGPDFPrintWhat flag. Default is IG_PDF_PRINT_DOCUMENT.
printWhatAnnot	Combination of enumIGPDFPrintWhatAnnot flags which extend printWhat to enable Pro product behavior.
emitPS	If true, emit a PostScript file. The default value is true.
psLevel	PostScript level: 1, 2 or 3. The default value is 2.
outputType	Print PostScript or EPS with or without a preview.
incBaseFonts	Embed the base fonts. The default value is IG_PDF_INCLUDE_NEVER.
incEmbeddedFonts	Embed fonts that are embedded in the PDF file. This overrides the **incType1Fonts**, **incTrueTypeFonts**, and **incCIDFonts** fields. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incType1Fonts	Embed Type 1 fonts. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incType3Fonts	Embed Type 3 fonts. The default value is IG_PDF_INCLUDE_ON_EVERY_PAGE. **NOTE:** This parameter must always be set to IG_PDF_INCLUDE_ON_EVERY_PAGE. PDF files exist with Type 3 fonts that contain different encodings on different pages.
incTrueTypeFonts	Embed TrueType fonts. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incCIDFonts	Embed CID fonts. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incProcsets	Include Procsets in the file. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.

incOtherResources	Include all other types of resources in the file. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
fontPerDocVM	Amount of VM available for font downloading at the document level. Ignored if <=0. **NOTE:** This must be set to 0 for the toolkit; it is only used by the viewer.
emitShowpage	Emit save and restore showpage in PostScript files. The default value is true.
emitTTFontsFirst	Emit TrueType fonts before any other fonts. The default value is false.
setPageSize	(PostScript level 2 only) Set the page size on each page. Use the media box for outputting to PostScript files, use the crop box for EPS files. Default is false.
emitDSC	Write DSC (Document Structuring Conventions) comments. The default value is true.
setupProcsets	If procsets are included, also include init/term code. The default value is true. **NOTE:** This must be set to true.
emitColorSeps	Emit images for Level-1 separations. The default value is false.
binaryOK	True if binary data is permitted in the PostScript file, false otherwise. The default value is true.
useSubFileDecode	Add SubFileDecode filter to work around stream problems. The default value is false.
emitRawData	True if add no unnecessary filters when emitting image data, false otherwise. The default value is true.
TTasT42	If including TrueType fonts, convert to Type 42 fonts instead of Type 1 fonts. The default value is false.
scale	Document-wide scale factor. 100.0 = full size. The default value is 100.
emitExternalStreamRef	If an Image resource uses an external stream, emit code that points to the external file. The default value is false. **NOTE:** This must be set to false.
emitHalftones	Preserve any halftone screening in the PDF file. The default value is false.
emitPSXObjects	Emit PostScript XObjects into the PostScript stream. The default value is false.
centerCropBox	True if CropBox output is centered on the page when the CropBox < MediaBox, false otherwise. The default value is true.
emitSeparableImagesOnly	If emitting EPS, include only CMYK and gray images.

emitDeviceExtGState	When emitting the extended graphics state, include the device-dependent parameters (overprint, black generation, undercolor removal, transfer, halftone, halftone phase, smoothness, flatness, rendering intent) in addition to the device-independent parameters (font, line width, line cap, line join, miter limit, dash pattern). If this flag is false, only the device-independent parameters will be emitted. This flag overrides **emitHalftones**; if this is false, then halftones are not emitted. The default value is true.
boundingBox	If all zeroes, is ignored. Otherwise, is used for %BoundingBox DSC comment and in centerCropBox calculations and for setpagedevice. The default value is [0 0 0 0].
useFontAliasNames	Used when printing with system fonts. The default value is false.
emitPageRotation	Emit a concat at the beginning of each page so that the page is properly rotated. Used when emitting EPS. The default value is false.
reverse	If set to true, reverse the order of page output.
tCropBox	Temporary crop box to represent selected region.
emitPageClip	Set to true to emit page clip.
emitTransfer	Set to true to emit transfer.
emitBG	Set to true to emit black generation.
emitUCR	Set to true to emit undercolor removal.
farEastFontOpt	Far East font option. Currently not used. Set to 0.
suppressCJKSubstitution	If true, do not do CJK substitution on the printer.
suppressCSA	If true, don't emit CSAs for 4 component (CMYK) colors.
hostBased	For separator, do host-based color management.
hostBasedOutputCS	The output color space when hostBased color management is true.
duplex	Currently not used. Set to 0.
doTiling	Whether to tile none, all, or only large pages
tileInfo	If non-NULL, tiling is desired with these parameters.
rotate	Enable the auto-rotating behavior from past versions of Acrobat.
hostBasedCM	Host base color management. Default: false, do CSA generation for profiles instead of converting all colors on the host.
destProfile	If hostBaseCM color management is true, use this profile.

destCSAtom	An atom representing the device color space (DeviceGray, DeviceRGB, etc.).
saveVM	True means try to save VM when printing to PostScript.
doOPP	If true, do the overprint preview operation.
suppressOPPWhenNoSpots	When true, suppress OPP for pages that do not contain spot colors
optimizeForSpeed	If true, do it fast; false means PostScript code must be page independent. If set to true, font downloads are forced from IG_PDF_INCLUDE_ON_EVERY_PAGE to IG_PDF_INCLUDE_ONCE_PER_DOC.
brokenCRDs	If true, don't set rendering intent in PostScript stream due to broken non-default CRDs.
useMaxVM	If true, store all possible resources in VM.
lastWidth	Used when setPageSize is true to prevent unneeded setpagedevice calls.
lastHeight	Used when setPageSize is true to prevent unneeded setpagedevice calls.
bitmapResolution	DPI for bitmaps. Default is 300.
gradientResolution	DPI for gradients interior to the object (not edges). Can generally be lower than the bitmapResolution. Default is 150.
transparencyQuality	The transparency level. Range is 1-100.
ocContext	The optional-content context to use for visibility state information, or NULL to use the document's current states in the default context.
applyOCGPrintOverrides	When true, apply print-specific visibility state settings from the optional-content group.
useFullResolutionJP2KData	Whether to use the maximum available JPEG2000 resolution.
emitInRipSeps	When true, requests that separations, one sheet per ink, be generated in the RIP (printer).
whichMarks	Page mark indication. A bit-wise OR of the enumIGPDFPageMarkFlags values.
westernMarksStyle	When true, use western style for page marks.
doProofing	When true, print using proofing settings.
proofProfile	Description string for the proofing profile.
inkBlack	Proofing settings: simulate ink black.

paperWhite	Proofing settings: simulate paper white.
useExecForm	When true, emit execform calls when emitting Form XObjects.
flattenInfo	A structure containing parameters that control tile flattening.
negative	When true, invert the plate.
mirrorprint	PostScript mirroring attribute. Currently not used.
numCollatedCopies	Enables collation for viewer.
emitFlatness	Set to true to emit flatness.
trapType	Specifies trap type.
TTasCIDT2	Set to true to emit TrueType fonts as CIDType2 instead of as CIDFontType0.
markStyle	Specify the style to use for page marks.
lineWidth	Line width to use for printer marks.
macQDPrinter	Set to true if the printer is a Mac QuickDraw printer.

Definition:

This structure indicates how a document should be printed.

NOTE: All fields in the AT_PDF_PRINTPARAMS structure apply to PostScript file creation. AT_PDF_PRINTPARAMS are ignored when printing to non-PostScript devices.

AT_PDF_SECURITYDATA

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDF_SECURITYDATA
{
    DWORD size;
    AT_PDF_BOOL newUserPW;
    AT_PDF_BOOL hasUserPW;
    CHAR userPW[256];
    AT_PDF_BOOL newOwnerPW;
    AT_PDF_BOOL hasOwnerPW;
    CHAR ownerPW[256];
```

```
    UINT perms;
    LONG keyLength;
    LONG revision;
    AT_PDF_BOOL encryptMetadata;
    LONG encryptMethod;
    AT_PDF_BOOL encryptAttachmentsOnly;

} AT_PDF_SECURITYDATA;
typedef AT_PDF_SECURITYDATA FAR* LPAT_PDF_SECURITYDATA;
```

Arguments:

DWORD	size	Size of this stucture.
AT_PDF_BOOL	newUserPW	True if the user password should be changed.
AT_PDF_BOOL	hasUserPW	True if there is a user password.
CHAR	userPW[256]	The user password string.
AT_PDF_BOOL	newOwnerPW	True if the owner password should be changed, false otherwise.
AT_PDF_BOOL	hasOwnerPW	True if an owner password is provided, false otherwise.
CHAR	ownerPW[256]	The owner password string.
UINT	perms	Permissions to allow. An OR of the enumIGPDFPermsFlags values.
LONG	keyLength	Encryption key length in byte.
LONG	revision	Indicates /R value.
AT_PDF_BOOL	encryptMetadata	Flag that indicates whether document metadata will be encrypted.
LONG	encryptMethod	Method of encryption for filters to use. One of the enumIGPDFStdSecurityMethod values.
AT_PDF_BOOL	encryptAttachmentsOnly	Flag to indicate that only Attachments are encrypted - encryptMetadata and encryptAttachmentsOnly cannot both be true.

Description:

This structure describes the data for the standard security handler.

NOTE: The password strings in PDF are padded or truncated to exactly 32 bytes. If the password string is more than 32 bytes long, used only its first 32 bytes; if it is less than 32 bytes long, it padded by appending the required number of additional

bytes from the beginning of the following padding string: <28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

AT_PDE_FONTATTRS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_FONTATTRS
{
   HIG_PDF_ATOM name;
   HIG_PDF_ATOM type;
   HIG_PDF_ATOM charSet;
   HIG_PDF_ATOM encoding;
   UINT   flags;
   AT_PDF_FIXEDRECT fontBBox;
   SHORT  missingWidth;
   SHORT  stemV;
   SHORT  stemH;
   SHORT  capHeight;
   SHORT  xHeight;
   SHORT  ascent;
   SHORT  descent;
   SHORT  leading;
   SHORT  maxWidth;
   SHORT  avgWidth;
   SHORT  italicAngle;
   HIG_PDF_ATOM cidFontType;
   SHORT     wMode;
   HIG_PDF_ATOM psName;
   HIG_PDF_ATOM platformName;
   HIG_PDF_ATOM lang;
   HIG_PDF_ATOM registry;
   HIG_PDF_ATOM ordering;
   LONG   supplement;
   LONG   cantEmbed;
   HIG_PDF_ATOM deltaEncoding;
   UINT   protection;
   LONG   packageType;
} AT_PDE_FONTATTRS;
typedef AT_PDE_FONTATTRS FAR* LPAT_PDE_FONTATTRS;
```

Arguments:

HIG_PDF_ATOM	name	A HIG_PDF_ATOM for font name, as in "Times-Roman." Corresponds to the BaseFont key in the font dictionary of a PDF file (see Section 5.6.3 in the PDF Reference).
HIG_PDF_ATOM	type	A HIG_PDF_ATOM for font type, corresponding to the Subtype key in a font dictionary. May be "Type1," "TrueType," "MMType1," or "Type0."
HIG_PDF_ATOM	charSet	A HIG_PDF_ATOM for "Roman" or IG_PDF_ATOM_NULL. If "Roman," the characters must be a subset of the Adobe Standard Roman Character Set.
HIG_PDF_ATOM	encoding	An HIG_PDF_ATOM for font encoding. May be MacRomanEncoding, WinAnsiEncoding, or IG_PDF_ATOM_NULL. In the case of IG_PDF_ATOM_NULL, call PDSysFontGetEncoding to get more information about the encoding.
UINT	flags	Desired font flags, one or more of Font Flags. Use IG_PDF_SCRIPT, etc. to get flags.
AT_PDF_FIXEDRECT	fontBBox	Font bounding box in 1000 EM units
SHORT	missingWidth	Width of missing character (.notdef)
SHORT	stemV	Vertical stem width
SHORT	stemH	Horizontal stem width
SHORT	capHeight	Capital height
SHORT	xHeight	X height
SHORT	ascent	Max ascender height
SHORT	descent	Max descender depth
SHORT	leading	Additional leading between lines
SHORT	maxWidth	Maximum character width
SHORT	avgWidth	Average character width
SHORT	italicAngle	Italic angle in degrees, if any
HIG_PDF_ATOM	cidFontType	CIDFontType0 or CIDFontType2
SHORT	wMode	Writing mode. Must be one of 0 for horizontal writing or 1 for vertical writing
HIG_PDF_ATOM	psName	HIG_PDF_ATOM representing the PostScript name of a TrueType font.
HIG_PDF_ATOM	platformName	The platform name

HIG_PDF_ATOM	lang	HIG_PDF_ATOM representing the ISO 639 language code. These are available from http://www.iso.ch.
HIG_PDF_ATOM	registry	HIG_PDF_ATOM representing the CIDFont's Registry information, as in "gAdobe-Japan".
HIG_PDF_ATOM	ordering	HIG_PDF_ATOM representing the CIDFont's Ordering information, for example, "g1".
LONG	supplement	The SystemSupplement field from the CIDFont.
LONG	cantEmbed	A non-zero value means the font can't be embedded.
HIG_PDF_ATOM	deltaEncoding	The name of the base encoding; that is, the BaseEncoding entry in an encoding dictionary (see section 5.5.5 of the PDF Reference). The Differences entry of the encoding dictionary describes differences (deltas) from the base encoding.
UINT	protection	protection Allows setting one of the following bits to disable font embedding: IG_PDE_FONT_NO_EMBEDDING = 1: font should not be embedded. IG_PDE_FONT_NO_EDITABLE_EMBEDDING = 2: font should not be embedded for editing purposes.
LONG	packageType	enumIGPDFSysFontPackageType

Description:

Attributes for HIG_PDE_FONT and HIG_PDF_SYSFONT.

AT_PDE_COLORVALUE

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORVALUE
{
   AT_PDF_FIXED color[7];
   HIG_PDE_OBJECT colorObj2;
   HIG_PDE_OBJECT colorObj;

} AT_PDE_COLORVALUE;
typedef AT_PDE_COLORVALUE FAR* LPAT_PDE_COLORVALUE;
```

Arguments:

AT_PDF_FIXED	color[7]	Color value components. For instance, a Gray color space has 1 component, an RGB color space has 3 components, a CMYK has 4 components, and so on.
HIG_PDE_OBJECT	colorObj2	For DeviceN color space
HIG_PDE_OBJECT	colorObj	For color spaces whose color values do not have numeric values, such as the Pattern and Separation color spaces.

Description:

Describes color value.

AT_PDE_COLORSPEC

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORSPEC
{
    HIG_PDE_COLORSPACE space;
    AT_PDE_COLORVALUE value;

} AT_PDE_COLORSPEC;
typedef AT_PDE_COLORSPEC FAR* LPAT_PDE_COLORSPEC;
```

Arguments:

HIG_PDE_COLORSPACE	space	The specified color space
AT_PDE_COLORVALUE	value	The color value.

Description:

Describes color specification.

AT_PDE_DASH

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_DASH
{
    AT_PDF_FIXED dashPhase;
```

```
    LONG dashLen;
    AT_PDF_FIXED dashes[11];

} AT_PDE_DASH;
typedef AT_PDE_DASH FAR* LPAT_PDE_DASH;
```

Arguments:

AT_PDF_FIXED	dashPhase	Dash phase. Phase is a number that specifies a distance in user space into the dash pattern at which to begin marking the path.
LONG	dashLen	Number of entries in the dash array, an element of the Border array.
AT_PDF_FIXED	dashes[11]	Dash array, which specifies distances in user space for the length of dashes and gaps.

Description:

Describes dash specification, as described in Table 4.8 in the PDF Reference. See Section 4.3.2 for more information on line dash patterns.

AT_PDE_GRAPHICSTATE

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_GRAPHICSTATE
{
    UINT wasSetFlags;
    AT_PDE_COLORSPEC fillColorSpec;
    AT_PDE_COLORSPEC strokeColorSpec;
    AT_PDE_DASH dash;
    AT_PDF_FIXED lineWidth;
    AT_PDF_FIXED miterLimit;
    AT_PDF_FIXED flatness;
    LONG lineCap;
    LONG lineJoin;
    HIG_PDF_ATOM renderIntent;
    HIG_PDE_OBJECT extGState;

} AT_PDE_GRAPHICSTATE;
typedef AT_PDE_GRAPHICSTATE FAR* LPAT_PDE_GRAPHICSTATE;
```

Arguments:

UINT	wasSetFlags	enumIGPDEGraphicStateWasSetFlags indicating if an attribute has been set. **NOTE:** Support for these flags is not complete. For compatibility, you should set them, but do not depend on reading their values back. The intended use is with XObject Forms to indicate whether the value is inherited or explicitly set.
AT_PDE_COLORSPEC	fillColorSpec	Fill color specification. The default value is DeviceGray, IG_PDF_FIXED_ZERO.
AT_PDE_COLORSPEC	strokeColorSpec	Stroke color specification. The default value is DeviceGray, IG_PDF_FIXED_ZERO.
AT_PDE_DASH	dash	Dash specification. The default value is [0, 0].
AT_PDF_FIXED	lineWidth	Line width, corresponding to the w (setlinewidth) operator. The default value is IG_PDF_FIXED_ONE.
AT_PDF_FIXED	miterLimit	Miter limit, corresponding to the M (setmiterlimit) operator. The default value is IG_PDF_FIXED_TEN.
AT_PDF_FIXED	flatness	Line flatness, corresponding to the i (setflat) operator. The default value is IG_PDF_FIXED_ZERO.
LONG	lineCap	Line cap style, corresponding to the J (setlinecap) operator. The default value is 0.
LONG	lineJoin	Line join style, corresponding to the j (setlinejoin) operator. The default value is 0.
HIG_PDF_ATOM	renderIntent	A color rendering intent, corresponding to the Intent key in the image dictionary. The default value is 0.
HIG_PDE_OBJECT	extGState	An extended graphics, corresponding to the gs operator. The default value is NULL.

Description:

Attributes of a PDE element or a PDE text sub-element.

AT_PDE_TEXTSTATE

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_TEXTSTATE
```

```
{
    UINT wasSetFlags;
    AT_PDF_FIXED charSpacing;
    AT_PDF_FIXED wordSpacing;
    LONG renderMode;
    AT_PDF_FIXED fontSize;
    AT_PDF_FIXED hScale;
    AT_PDF_FIXED textRise;

} AT_PDE_TEXTSTATE;
typedef AT_PDE_TEXTSTATE FAR* LPAT_PDE_TEXTSTATE;
```

Arguments:

UINT	wasSetFlags	enumIGPDEGraphicStateWasSetFlags indicating if an attribute has been set. **NOTE:** Support for these flags is not complete. For compatibility, you should set them, but do not depend on reading their values back. The intended use is with XObject Forms to indicate whether the value is inherited or explicitly set. **NOTE:** PDFEdit ignores the wasSetFlags flag, so you must initialize the AT_PDE_TEXTSTATE fields.
AT_PDF_FIXED	charSpacing	Character spacing was set, corresponding to the Tc operator.
AT_PDF_FIXED	wordSpacing	Word spacing, corresponding to the Tw operator.
LONG	renderMode	Text rendering mode, corresponding to the Tr operator.
AT_PDF_FIXED	fontSize	Default is 1.
AT_PDF_FIXED	hScale	Default=100 (==100%)
AT_PDF_FIXED	textRise	Specifies the distance, in text space units that are not scaled, to move the baseline up or down from its default location. See Section 5.2.6 in the PDF Reference.

Description:

Attributes of a PDE text element.

AT_PDF_SYSFONT_PLATDATA

Group: Subsidiary Type

Declaration (WINDOWS):

```
typedef struct tagAT_PDF_SYSFONT_PLATDATA
{
    DWORD size;
    LOGFONT *lf;
    DWORD fontPath;
    DWORD afmPath;

} AT_PDF_SYSFONT_PLATDATA;
typedef AT_PDF_SYSFONT_PLATDATA FAR* LPAT_PDF_SYSFONT_PLATDATA;
```

Arguments (WINDOWS):

DWORD	size	sizeof(AT_PDF_SYSFONT_PLATDATA)
LOGFONT	*lf	(Windows only) Windows LOGFONT structure defining font attributes.
DWORD	fontPath	Not used
DWORD	afmPath	Not used

Declaration (MAC):

```
typedef struct tagAT_PDF_SYSFONT_PLATDATA
{
    DWORD size;
    PDFSysFontFID fontID;
    PDFSysFontFStyle fontStyle;

} AT_PDF_SYSFONT_PLATDATA;
typedef AT_PDF_SYSFONT_PLATDATA FAR* LPAT_PDF_SYSFONT_PLATDATA;
```

Arguments (MAC):

DWORD	size	sizeof(AT_PDF_SYSFONT_PLATDATA)
PDFSysFontFID	fontID	FontID
PDFSysFontFID	fontStyle	Style value within that Font - default = 0

Declaration (UNIX):

```
typedef struct tagAT_PDF_SYSFONT_PLATDATA
{
    DWORD size;
```

```
    DWORD fontPath;
    DWORD afmPath;

} AT_PDF_SYSFONT_PLATDATA;
typedef AT_PDF_SYSFONT_PLATDATA FAR* LPAT_PDF_SYSFONT_PLATDATA;
```

Arguments (UNIX):

DWORD	size	sizeof(AT_PDF_SYSFONT_PLATDATA)
DWORD	fontPath	Not used
DWORD	afmPath	Not used

Description:

SysFont platform specific data.

AT_PDE_FONT_INFO

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_FONT_INFO
{
    HIG_PDF_ATOM name;
    HIG_PDF_ATOM type;
    HIG_PDF_ATOM charSet;
    HIG_PDF_ATOM encoding;
    SHORT wMode;

} AT_PDE_FONT_INFO;
typedef AT_PDE_FONT_INFO FAR* LPAT_PDE_FONT_INFO;
```

Arguments:

HIG_PDF_ATOM	name	HIG_PDF_ATOM for font name, as in "Times-Roman."
HIG_PDF_ATOM	type	HIG_PDF_ATOM for font type, "Type 1," "TrueType," and so on.
HIG_PDF_ATOM	charSet	HIG_PDF_ATOM for "Roman" or IG_PDF_ATOM_NULL. If "Roman," the characters must be a subset of the Adobe Standard Roman Character Set.
HIG_PDF_ATOM	encoding	HIG_PDF_ATOM for font encoding, as in WinAnsiEncoding.
SHORT	wMode	Writing mode: 0 = horizontal; 1 = vertical.

Description:

PDE font information

AT_PDE_FONT_CREATEFROMSYSFONTPARAMS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_FONT_CREATEFROMSYSFONTPARAMS
{
   UINT structSize;
   UINT flags;
   HIG_PDF_ATOM snapshotName;
   AT_PDF_FIXED *mmDesignVec;
   LONG ctCodePage;
   HIG_PDF_ATOM encoding;

} AT_PDE_FONT_CREATEFROMSYSFONTPARAMS;
typedef AT_PDE_FONT_CREATEFROMSYSFONTPARAMS FAR*
LPAT_PDE_FONT_CREATEFROMSYSFONTPARAMS;
```

Arguments:

UINT	structSize	Size of the data structure. Must be set to sizeof(AT_PDE_FONT_CREATEFROMSYSFONTPARAMS).
UINT	flags	A bit mask of the enumIGPDEFontCreateFlags.
HIG_PDF_ATOM	snapshotName	The name of a multiple master snapshot. See PDF Reference for more information on snapshots.
AT_PDF_FIXED	*mmDesignVec	Pointer to multiple master font design vector.
LONG	ctCodePage	Used to select a specific code page supported by the font. When a non-zero code page is supplied, embedding must be turned on and the IG_PDE_FONT_ENCODE_BY_GID flag set.
HIG_PDF_ATOM	encoding	Used to specify which encoding to use with a CID font. Pass IG_PDF_ATOM_NULL to use the platform default.

Description:

Data structure used with PDE Font creation.

AT_PDE_COLORRANGE

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORRANGE
{
   float min;
   float max;
} AT_PDE_COLORRANGE;
typedef AT_PDE_COLORRANGE FAR* LPAT_PDE_COLORRANGE;
```

Arguments:

float	min	
float	max	

Description:

Contains color range

AT_PDE_XYZCOLOR

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_XYZCOLOR
{
    float x;
    float y;
    float z;

} AT_PDE_XYZCOLOR;
typedef AT_PDE_XYZCOLOR FAR* LPAT_PDE_XYZCOLOR;
```

Arguments:

float	x	
float	y	
float	z	

Description:

XYZ color data

AT_PDE_COLORDATA_CALGRAY

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_CALGRAY
{
    AT_PDE_XYZCOLOR whitePoint;
    AT_PDE_XYZCOLOR blackPoint;
    float gamma;
} AT_PDE_COLORDATA_CALGRAY;
typedef AT_PDE_COLORDATA_CALGRAY FAR*
LPAT_PDE_COLORDATA_CALGRAY;
```

Arguments:

AT_PDE_XYZCOLOR	whitePoint	White point
AT_PDE_XYZCOLOR	blackPoint	Black point
float	gamma	Gamma

Description:

Describes a CalGray color space.

Default calGray = {{0, 0, 0}, {0, 0, 0}, 1};

AT_PDE_COLORDATA_CALRGB

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_CALRGB
{
    AT_PDE_XYZCOLOR whitePoint;
    AT_PDE_XYZCOLOR blackPoint;
    float redGamma;
    float greenGamma;
    float blueGamma;
    float matrix[9];

} AT_PDE_COLORDATA_CALRGB;
typedef AT_PDE_COLORDATA_CALRGB FAR* LPAT_PDE_COLORDATA_CALRGB;
```

Arguments:

AT_PDE_XYZCOLOR	whitePoint	White point
AT_PDE_XYZCOLOR	blackPoint	Black point
float	redGamma	Red gamma
float	greenGamma	Green gamma
float	blueGamma	Blue gamma
float	matrix[9]	Matrix

Description:

Describes a CalRGB color space.

Default calRGB = {{0, 0, 0}, {0, 0, 0}, 1, 1, 1, {1, 0, 0, 0, 1, 0, 0, 0, 1}};

AT_PDE_COLORDATA_LAB

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_LAB
{
    AT_PDE_XYZCOLOR whitePoint;
    AT_PDE_XYZCOLOR blackPoint;
    AT_PDE_COLORRANGE rangeA, rangeB;

} AT_PDE_COLORDATA_LAB;
typedef AT_PDE_COLORDATA_LAB FAR* LPAT_PDE_COLORDATA_LAB;
```

Arguments:

AT_PDE_XYZCOLOR	whitePoint	White point
AT_PDE_XYZCOLOR	blackPoint	Black point
AT_PDE_COLORRANGE	rangeA, rangeB	Color ranges

Description:

Describes a L*a*b* color space.

Default lab = {{0, 0, 0}, {0, 0, 0}, {-100, 100}, {-100, 100}};

AT_PDE_COLORDATA_ICCBASED

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_ICCBASED
{
    DWORD size;
    HIG_PDF_STREAM iccstream;
    UINT nComps;
    HIG_PDE_COLORSPACE altCs;

} AT_PDE_COLORDATA_ICCBASED;
typedef AT_PDE_COLORDATA_ICCBASED FAR*
LPAT_PDE_COLORDATA_ICCBASED;
```

Arguments:

DWORD	size	size = sizeof(AT_PDE_COLORDATA_ICCBASED)
HIG_PDF_STREAM	iccstream	stream containing ICC Profile
UINT	nComps	Number of color components (1, 3, or 4)
HIG_PDE_COLORSPACE	altCs	Alternate ColorSpace (optional)

Description:

ICC based color space

AT_PDE_COLORDATA_INDEXED

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_INDEXED
{
    DWORD size;
    HIG_PDE_COLORSPACE baseCs;
    WORD hival;
    LPBYTE lookup;
    UINT lookupLen;

} AT_PDE_COLORDATA_INDEXED;
typedef AT_PDE_COLORDATA_INDEXED FAR*
LPAT_PDE_COLORDATA_INDEXED;
```

Arguments:

DWORD	size	size = sizeof(AT_PDE_COLORDATA_INDEXED)
HIG_PDE_COLORSPACE	baseCs	base colorspace
WORD	hival	highest color value
LPBYTE	lookup	Indexed color lookup data.
UINT	lookupLen	number of bytes in lookup data

Description:

Indexed color space

AT_PDE_COLORDATA_SEPARATION

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_SEPARATION
{
    DWORD size;
    HIG_PDF_ATOM name;
    HIG_PDE_COLORSPACE alt;
    HIG_PDF_BASOBJ tintTransform;

} AT_PDE_COLORDATA_SEPARATION;
typedef AT_PDE_COLORDATA_SEPARATION FAR*
LPAT_PDE_COLORDATA_SEPARATION;
```

Arguments:

DWORD	size	size = sizeof(AT_PDE_COLORDATA_SEPARATION)
HIG_PDF_ATOM	name	name of separation or colorant
HIG_PDE_COLORSPACE	alt	Alternative colorspace
HIG_PDF_BASOBJ	tintTransform	The tintTransform dictionary or function. See Section 4.5.5 in the PDF Reference for more information.

Description:

Separation color space

AT_PDE_COLORDATA_DEVICEN

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_DEVICEN
{
    DWORD size;
    HIG_PDF_ATOM *names;
    UINT nNames;
    HIG_PDE_COLORSPACE alt;
    HIG_PDF_BASOBJ tintTransform;

} AT_PDE_COLORDATA_DEVICEN;
typedef AT_PDE_COLORDATA_DEVICEN FAR*
LPAT_PDE_COLORDATA_DEVICEN;
```

Arguments:

DWORD	size	size = sizeof(AT_PDE_COLORDATA_DEVICEN)
HIG_PDF_ATOM	*names	names of colorants
UINT	nNames	number of colorants
HIG_PDE_COLORSPACE	alt	Alternative colorspace
HIG_PDF_BASOBJ	tintTransform	The tintTransform dictionary or function. See Section 4.5.5 in the PDF Reference for more information.

Description:

DeviceN color space.

AT_PDE_COLORDATA

Group: Subsidiary Type

Declaration:

```
typedef union tagAT_PDE_COLORDATA
{
   AT_PDE_COLORDATA_CALGRAY *calGray;
   AT_PDE_COLORDATA_CALRGB *calRGB;
   AT_PDE_COLORDATA_LAB *lab;
   AT_PDE_COLORDATA_ICCBASED *icc;
   AT_PDE_COLORDATA_INDEXED *indexed;
   HIG_PDE_COLORSPACE patternbase;
   AT_PDE_COLORDATA_SEPARATION *sep;
   AT_PDE_COLORDATA_DEVICEN *devn;

} AT_PDE_COLORDATA;
typedef AT_PDE_COLORDATA FAR* LPAT_PDE_COLORDATA;
```

Arguments:

AT_PDE_COLORDATA_CALGRAY	*calGray	
AT_PDE_COLORDATA_CALRGB	*calRGB	
AT_PDE_COLORDATA_LAB	*lab	
AT_PDE_COLORDATA_ICCBASED	*icc	
AT_PDE_COLORDATA_INDEXED	*indexed	
HIG_PDE_COLORSPACE	patternbase	
AT_PDE_COLORDATA_SEPARATION	*sep	
AT_PDE_COLORDATA_DEVICEN	*devn	

Description:

Fill in one of the following members of this union then pass this data to PDE color space creation routine. Please see section 7.10 of the PDF Reference Manual for information on color spaces.

AT_PDE_PSATTRS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_PSATTRS
{
    UINT flags;

} AT_PDE_PSATTRS;
typedef AT_PDE_PSATTRS FAR* LPAT_PDE_PSATTRS;
```

Arguments:

UINT	flags	IG_PDE_PS_INLINE

Description:

Attributes of a PDE PostScript object

AT_PDE_IMAGEATTRS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_IMAGEATTRS
{
    UINT flags;
    LONG width;
    LONG height;
    LONG bitsPerComponent;
    AT_PDF_FIXED decode[8];
    HIG_PDF_ATOM intent;

} AT_PDE_IMAGEATTRS;
typedef AT_PDE_IMAGEATTRS FAR* LPAT_PDE_IMAGEATTRS;
```

Arguments:

UINT	flags	enumIGPDEImageAttrFlags indicating image attributes.
LONG	width	Width of the image, corresponding to the Width key in the image dictionary.
LONG	height	Height of the image, corresponding to the Height key in the image dictionary.
LONG	bitsPerComponent	Number of bits used to represent each color component in the image, corresponding to the BitsPerComponent key in the image dictionary.
AT_PDF_FIXED	decode[8]	An array of numbers specifying the mapping from sample values in the image to values appropriate for the current color space. These values correspond to the Decode key in the image dictionary.
HIG_PDF_ATOM	intent	Color rendering intent, corresponding to the Intent key in the image dictionary.

Description:

Attributes of a PDE Image object

AT_PDE_FILTERSPEC

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_FILTERSPEC
{
   HIG_PDF_BASOBJ decodeParms;
   HIG_PDF_BASOBJ encodeParms;
   HIG_PDF_ATOM name;

} AT_PDE_FILTERSPEC;
typedef AT_PDE_FILTERSPEC FAR* LPAT_PDE_FILTERSPEC;
```

Arguments:

HIG_PDF_BASOBJ	decodeParms	Parameters used by the decoding filters specified with the Filter key. Corresponds to the DecodeParms key in the stream dictionary. Must be set to NULL if AT_PDE_FILTERSPEC is specified but no decode parameters are specified. This can be done by zeroing the unused decode params. Required decode params for DCTDecode are Columns, Rows, and Colors.
HIG_PDF_BASOBJ	encodeParms	Parameters used when encoding the stream. Required for DCTDecode filter; optional for other filters. Must be set to NULL if AT_PDE_FILTERSPEC is specified but no encode parameters are specified. This can be done by zeroing the unused encode params.
HIG_PDF_ATOM	name	Filter name. Supported filters are: ASCIIHexDecode, ASCII85Decode, LZWDecode, DCTDecode, CCITTFaxDecode, RunLengthDecode, and FlateDecode.

Description:

Filter element in a filter array.

AT_PDE_FILTERARRAY

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_FILTERARRAY
{
   LONG numFilters;
```

```
    AT_PDE_FILTERSPEC spec[2];

} AT_PDE_FILTERARRAY;
typedef AT_PDE_FILTERARRAY FAR* LPAT_PDE_FILTERARRAY;
```

Arguments:

LONG	numFilters	Number of filters in the array.
AT_PDE_FILTERSPEC	spec[2]	Variable length array of filter spec.

Description:

Array of filter specifications usually containing two or less filters: encoding and/or compression.

AT_PDE_CONTENTATTRS

Group: Subsidiary Type

Declaration:

```
typedef struct tagAT_PDE_CONTENTATTRS
{
    UINT flags;
    AT_PDF_FIXED cacheDevice[8];
    LONG formType;
    AT_PDF_FIXEDRECT bbox;
    AT_PDF_FIXEDMATRIX matrix;
    HIG_PDF_BASOBJ XUID;

} AT_PDE_CONTENTATTRS;
typedef AT_PDE_CONTENTATTRS FAR* LPAT_PDE_CONTENTATTRS;
```

Arguments:

UINT	flags	enumIGPDEContentFlags
AT_PDF_FIXED	cacheDevice[8]	CharProc attributes If flags has IG_PDE_SET_CACHE_DEVICE set, the first 6 cache device values contain the operands for the d1 (setcachdevice) page operator. If flags has IG_PDE_SET_CHAR_WIDTH set, cacheDevice contains 2 charwidth values.
LONG	formType	Form attributes Only used if HIG_PDE_CONTENT contains a Form XObject. Corresponds to FormType key in the XObject Form attributes dictionary.
AT_PDF_FIXEDRECT	bbox	Only used if HIG_PDE_CONTENT contains a Form. Bounding box of the HIG_PDE_CONTENT object. Corresponds to BBox key in the XObject Form attributes dictionary.
AT_PDF_FIXEDMATRIX	matrix	Only used if HIG_PDE_CONTENT contains a Form. Transformation matrix for the HIG_PDE_CONTENT object. Corresponds to Matrix key in the XObject Form attributes dictionary.
HIG_PDF_BASOBJ	XUID	Only used if HIG_PDE_CONTENT contains a Form. The form's XUID, an ID that uniquely identifies the form. Corresponds to XUID key in the XObject Form attributes dictionary.

Description:

Attributes of a PDE Content object

Enumerations

This section represents ImageGear PDF component enumerations including their meaning and values.

enumIGPDFBookmarkFlags

Group: Enumeration

Description:

Represents PDF bookmark flags.

Values:

IG_PDF_BOOKMARK_FONT_ITALIC	1	Italic font
IG_PDF_BOOKMARK_FONT_BOLD	2	Bold font

enumIGPDFCharset

enumIGPDFDestinationType

IG_PDF_CHARSET_UNKNOWN	0	The font does not use Adobe Standard Encoding.
IG_PDF_CHARSET_ROMAN	1	The font uses Adobe Standard encoding. This is determined by the "Uses Adobe Standard Encoding" bit in the font descriptor.
IG_PDF_CHARSET_EXPERT	2	Currently unused.
IG_PDF_CHARSET_LAST		

Group: Enumeration

Description:

Represents the types of PDF object destination.

Values:

IG_PDF_DEST_INVALID	0	Invalid destination
IG_PDF_DEST_EXPLICIT	1	Explicit destination
IG_PDF_DEST_NAMED	2	Named destination

enumIGPDFFixedValues

Group: Enumeration

Description:

PDF Fixed value. A variety of predefined fixed-point constants.

Values:

IG_PDF_FIXED_ZERO	((LONG) 0x00000000L)	0
IG_PDF_FIXED_HUNDREDTH	((LONG) 0x0000028FL)	1/100
IG_PDF_FIXED_SIXTEENTH	((LONG) 0x00001000L)	1/16
IG_PDF_FIXED_TWELFTH	((LONG) 0x00001555L)	1/12
IG_PDF_FIXED_TENTH	((LONG) 0x00001999L)	1/10
IG_PDF_FIXED_EIGHTH	((LONG) 0x00002000L)	1/8
IG_PDF_FIXED_QUARTER	((LONG) 0x00004000L)	1/4
IG_PDF_FIXED_THIRD	((LONG) 0x00005555L)	1/3
IG_PDF_FIXED_HALF	((LONG) 0x00008000L)	1/2
IG_PDF_FIXED_TWOTHIRDS	((LONG) 0x0000AAAAL)	2/3
IG_PDF_FIXED_THREEQUARTERS	((LONG) 0x0000C000L)	3/4
IG_PDF_FIXED_PI4	((LONG) 0x0000c910L)	PI/4
IG_PDF_FIXED_SEVENEIGHTS	((LONG) 0x0000E000L)	7/8
IG_PDF_FIXED_ONE1	((LONG) 0x0000ffffL)	-1
IG_PDF_FIXED_ONE	((LONG) 0x00010000L)	1
IG_PDF_FIXED_PI2	((LONG) 0x00019220L)	PI/2
IG_PDF_FIXED_GOLDEN	((LONG) 0x00019e37L)	Golden
IG_PDF_FIXED_TEN	((LONG) 0x000A0000L)	10
IG_PDF_FIXED_MAX	((LONG) 0x7FFFFFFF)	Max fixed-point value
IG_PDF_FIXED_MIN	((LONG) 0x80000000)	Min fixed-point value

IG_PDF_FIXED_NEGATIVE_INFINITY	((LONG) IG_PDF_FIXED_MAX)	Negative fixed-point infinity
IG_PDF_FIXED_POSITIVE_INFINITY	((LONG) IG_PDF_FIXED_MIN)	Positive fixed-point infinity

enumIGPDEType

Group: Enumeration

Description:

Types of the editing objects.

Values:

IG_PDE_CONTENT	0	Content
IG_PDE_TEXT	1	Text
IG_PDE_PATH	2	Path
IG_PDE_IMAGE	3	Image
IG_PDE_FORM	4	Form
IG_PDE_POSTSCRIPT	5	PostScript
IG_PDE_XOBJECT	6	XObject
IG_PDE_CLIP	7	Clip
IG_PDE_FONT	8	Font
IG_PDE_COLORSPACE	9	ColorSpace
IG_PDE_GSTATE	10	Graphic State
IG_PDE_PLACE	11	Place
IG_PDE_CONTAINER	12	Container
IG_PDF_SYSFONT	13	System Font
IG_PDE_PATTERN	14	Pattern
IG_PDE_DEVICENCOLORS	15	Device N Colors
IG_PDE_SHADING	16	Shading
IG_PDE_GROUP	17	Group
IG_PDE_UNKNOWN	18	Unknown
IG_PDE_BEGIN_CONTAINER	19	Begin Container
IG_PDE_END_CONTAINER	20	End Container
IG_PDE_BEGIN_GROUP	21	Begin Group
IG_PDE_END_GROUP	22	End Group
IG_PDE_XGROUP	23	XGroup
IG_PDE_SOFTMASK	24	SoftMask
IG_PDF_SYSENCODING	25	System Encoding
IG_PDE_DOC	26	Document

IG_PDE_PAGE	27	Page
IG_PDE_READER	28	Reader
IG_PDE_WRITER	29	Writer
IG_PDE_TEXTITEM	30	Text Item
IG_PDE_LASTTYPE	31	Last Type

enumIGPDEFontProtection

Group: Enumeration

Description:

Setting for disabling font embedding.

Values:

IG_PDE_FONT_NO_EMBEDDING	0x00000001	Flags for protection of AT_PDE_FONTATTRS - embedding is not allowed.
IG_PDE_FONT_NO_EDITABLE_EMBEDDING	0x00000002	Flags for protection of AT_PDE_FONTATTRS - editable embedding is not allowed.

enumIGPDETextFlags

Group: Enumeration

Description:

Flags used in the text API.

Values:

IG_PDE_TEXT_RUN	0x0001	Specifies an action for the run
IG_PDE_TEXT_CHAR	0x0002	Specifies an action for single character
IG_PDE_TEXT_PAGE_SPACE	0x0004	Specifies user space
IG_PDE_TEXT_GET_BOUNDS	0x0008	Specifies using the font descriptor's FontBBox

enumIGPDFPageNumber

Group: Enumeration

Description:

PageNumber specification.

Values:

IG_PDF_BEFORE_FIRST_PAGE	-1	Specifies position before the first page. Usually used to insert first document page.
IG_PDF_LAST_PAGE	-2	Specifies the last page position. Usually used to insert last document page.

enumIGPDEInsertElement

Group: Enumeration

Description:

Used for inserting a PDE element into the content.

Values:

IG_PDE_BEFORE_FIRST	((LONG) -1)	Specifies position before the first element. Usually used to insert first content element.
IG_PDE_AFTER_LAST	(IG_PDF_FIXED_ MAX - 1)	Specifies the last element position. Usually used to insert last content element.

enumIGPDFFontFlags

Group: Enumeration

Description:

Font flags. Constants that indicate a font's attributes (fixed width, roman or symbolic, sans serif, and so forth).

Values:

IG_PDF_FIXED_WIDTH	0x00000001	All glyphs in the font are the same width.
IG_PDF_SERIF	0x00000002	The font is a serif font.
IG_PDF_PI	0x00000004	The font is a symbolic (pi) font.
IG_PDF_SCRIPT	0x00000008	The font is a script font.
IG_PDF_STD_ENCODING	0x00000020	The font uses standard encoding.
IG_PDF_ITALIC	0x00000040	The font is an italic font.
IG_PDF_ALL_CAP	0x00010000	The font is an all-caps font.
IG_PDF_SMALL_CAP	0x00020000	The font is a small caps font.
IG_PDF_FORCE_BOLD	0x00040000	Force bold characters to draw bold even at small point sizes.

enumIGPDEFontCreateFlags

Group: Enumeration

Description:

Flags for PDE font creation routine. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags.

Values:

IG_PDE_FONT_CREATE_NOT_ALLOWED	-1	Creation is not allowed. Usually returns by SysFont's "getCreateFlags" when the combination of SysFont and SysEncoding is not allowed.
IG_PDE_FONT_CREATE_EMBEDDED	0x0001	Embed the font. Create an embedded font. By itself, this will not subset the font.
IG_PDE_FONT_WILL_SUBSET	0x0002	Subset the font. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags. You must call "subsetNow" to actually subset the font. Both embedding and sub-setting a font creates a CFF font.
IG_PDE_FONT_DO_NOT_EMBED	0x0004	Do not embed the font. You cannot set both this and the IG_PDE_FONT_WILL_SUBSET flags. Nor can you set IG_PDE_FONT_CREATE_EMBEDDED.
IG_PDE_FONT_ENCODE_BY_GID	0x0008	Create a CIDFont with identity (GID) encoding.
IG_PDE_FONT_DEFER_WIDTHS	0x0010	Wait to get widths until later (affects Type0 fonts only).
IG_PDE_FONT_CREATE_SUBSET	0x0002	Subset the font. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags. You must call "subsetNow" to actually subset the font. Both embedding and sub-setting a font creates a CFF font.
IG_PDE_FONT_CREATE_GID_OVERRIDE	0x0020	The library will convert cp to gid with identity embedded
IG_PDE_FONT_CREATE_TO_UNICODE	0x0040	Create ToUnicode cmap
IG_PDE_FONT_CREATE_ALL_WIDTHS	0x0080	Supply entire widths table (affects Type0 fonts only)

enumIGPDEFontCreateNeedFlags

Group: Enumeration

Description:

Flags for PDE Font CreateNeedFlags.

Values:

IG_PDE_FONT_CREATE_NEED_WIDTHS	0x00010000	Need to create width
IG_PDE_FONT_CREATE_NEED_TO_UNICODE	0x00020000	Need to create ToUnicode stream
IG_PDE_FONT_CREATE_NEED_EMBED	0x00040000	Need to embed it

enumIGPDFSysFontPackageType

Group: Enumeration

Description:

Flags for packageType of AT_PDE_FONTATTRS/IGPDEFontAttrs.

Values:

IG_PDF_SYSFONT_UNKNOWN	0	Unknown
IG_PDF_SYSFONT_TYPE1	1	Type1
IG_PDF_SYSFONT_TRUETYPE	2	TrueType
IG_PDF_SYSFONT_CID	3	CID
IG_PDF_SYSFONT_ATC	4	ATC
IG_PDF_SYSFONT_OCF	5	OCF
IG_PDF_SYSFONT_OPENTYPE_CFF	6	OpenType CFF
IG_PDF_SYSFONT_OPENTYPE_CID	7	OpenType CID
IG_PDF_SYSFONT_OPENTYPE_TT	8	OpenType TT

enumIGPDFSysFontMatchFlags

Group: Enumeration

Description:

Font matching flags for SysFont find routine.

Values:

IG_PDF_SYSFONT_MATCH_NAME_AND_CHARSET	0x0001	Match the font name and character set.
IG_PDF_SYSFONT_MATCH_FONT_TYPE	0x0002	Match the font type
IG_PDF_SYSFONT_MATCH_WRITING_MODE	0x0004	Match the writing mode, that is, horizontal or vertical.

enumIGPDEGraphicStateWasSetFlags

Group: Enumeration

Description:

Structure describing the graphics state that was set.

Values:

IG_PDE_FILL_CSPACE_WAS_SET	0x0001	A fill color space was set, corresponding to the cs (setcolorspace) operator.
IG_PDE_FILL_CVALUE_WAS_SET	0x0002	A color fill value was set, corresponding to the sc (setcolor) operator.
IG_PDE_STROKE_CSPACE_WAS_SET	0x0004	A color space stroke value was set, corresponding to the CS (setcolorspace) operator.
IG_PDE_STROKE_CVALUE_WAS_SET	0x0008	A color stroke value was set, corresponding to the SC (setcolor) operator.
IG_PDE_DASH_WAS_SET	0x0010	A dash specification was set, corresponding to the d (setdash) operator.
IG_PDE_LINE_WIDTH_WAS_SET	0x0020	The line width was set, corresponding to the w (setlinewidth) operator.
IG_PDE_MITER_LIMIT_WAS_SET	0x0040	The miter limit was set, corresponding to the M (setmiterlimit) operator.
IG_PDE_FLATNESS_WAS_SET	0x0080	Line flatness was set, corresponding to the i (setflat) operator.
IG_PDE_LINE_CAP_WAS_SET	0x0100	Line cap style was set, corresponding to the J (setlinecap) operator.
IG_PDE_LINE_JOIN_WAS_SET	0x0200	Line join style was set, corresponding to the j (setlinejoin) operator.
IG_PDE_RENDER_INTENT_WAS_SET	0x0400	A color rendering intent was set, corresponding to the Intent key in the image dictionary.
IG_PDE_EXT_GSTATE_WAS_SET	0x0800	An extended graphics state was set, corresponding to the gs operator.

enumIGPDEImageDataFlags

Group: Enumeration

Description:

Image Data Flags.

Values:

AT_PDE_IMAGE_DATA_NOT_ENCODED	0x0000	Indicates filter is active; data is not encoded.
AT_PDE_IMAGE_ENCODED_DATA	0x0001	Indicates filter is active; data is encoded.

enumIGPDEXGroupCreateFlags

Group: Enumeration

Description:

Enumerated data type used to specify the type of transparency group to create.

Values:

IG_PDE_XGROUP_TYPE_TRANSPARENCY	0x0001	Creates a transparency XGroup object.

enumIGPDESoftMaskCreateFlags

Group: Enumeration

Description:

Flags for use with SoftMask create.

Values:

IG_PDE_SOFTMASK_TYPE_LUMINOSITY	0x0001	Specifies how the mask is to be computed.
IG_PDE_SOFTMASK_TYPE_ALPHA	0x0002	Specifies how the mask is to be computed.

enumIGPDEImageAttrFlags

Group: Enumeration

Description:

Flags for AT_PDE_IMAGEATTRS/IGPDEImageAttrs. See Section 4.8.4 in the PDF Reference for more information on image attributes.

Values:

IG_PDE_IMAGE_EXTERNAL	0x0001	Image is an XObject
IG_PDE_IMAGE_MASK	0x0002	Image is an imagemask
IG_PDE_IMAGE_INTERPOLATE	0x0004	Interpolate is true
IG_PDE_IMAGE_HAVE_DECODE	0x0008	We have a decode array
IG_PDE_IMAGE_INDEXED	0x0010	Uses an indexed color space
IG_PDE_IMAGE_MASKED_BY_POSITION	0x0020	Image has a Mask key containing an ImageMask stream
IG_PDE_IMAGE_MASKED_BY_COLOR	0x0040	Image has a Mask key containing an array of color values

enumIGPDEPSAttrFlags

Group: Enumeration

Description:

Flags for AT_PDE_PSATTRS/IGPDEPSAttrs.

Values:

IG_PDE_PS_INLINE	0	Inline PostScript

enumIGPDEElementCopyFlags

Group: Enumeration

Description:

Bitfield for PDE element copy.

Values:

IG_PDE_ELEMENT_COPY_FOR_CLIP	0x0001	Copied element does not need gstate or clip
IG_PDE_ELEMENT_COPY_CLIPPING	0x0002	Acquire the clip path and put it in the copied object

enumIGPDEContentGetResourceFlags

Group: Enumeration

Description:

Bit field for AT_PDE_CONTENTATTRS/IGPDEContentAttrs.

Values:

IG_PDE_GET_FONTS	0	Obtain font resources.
IG_PDE_GET_XOBJECTS	1	Obtain Xobject resources.
IG_PDE_GET_COLORSPACES	2	Obtain color space resources.

enumIGPDEContentFlags

Group: Enumeration

Description:

Bit field for AT_PDE_CONTENTATTRS/IGPDEContentAttrs.

Values:

IG_PDE_SET_CACHE_DEVICE	0x0001	if set, cacheDevice contains 6 cache device values
IG_PDE_SET_CHAR_WIDTH	0x0002	if set, cacheDevice contains 2 charwidth values
IG_PDE_FORM_MATRIX	0x0004	if set, formMatrix contains a valid matrix

enumIGPDFWordFlags

Group: Enumeration

Description:

Context flags.

Values:

IG_PDF_ORDER	0x2	Use PDF order for text enumeration
IG_PDF_XY_SORT	0x4	Use XY order for text enumeration

enumIGPDFWordFinderVersion

Group: Enumeration

Description:

WordFinder algorithm version.

Values:

IG_PDF_WF_LATEST_VERSION	0	The latest available version.
IG_PDF_WF_VERSION_2	2	Version used for Acrobat 3.x, 4.x.
IG_PDF_WF_VERSION_3	3	Available in Acrobat 5.0 without Accessibility enabled. Includes some improved word piecing algorithms.
IG_PDF_WF_VERSION_4	4	For Acrobat 5.0 with Accessibility enabled. Includes advanced word ordering algorithms in addition to improved word piecing algorithms.

enumIGPDFCodePages

Group: Enumeration

Description:

Code page character-mapping constants.

Values:

IG_PDF_CODEPAGE_WIN_EAST_EUROPEAN_ROMAN	1250	Windows code pages
IG_PDF_CODEPAGE_WIN_CYRILLIC	1251	Windows code pages
IG_PDF_CODEPAGE_WIN_GREEK	1253	Windows code pages
IG_PDF_CODEPAGE_WIN_TURKISH	1254	Windows code pages
IG_PDF_CODEPAGE_WIN_HEBREW	1255	Windows code pages
IG_PDF_CODEPAGE_WIN_ARABIC	1256	Windows code pages
IG_PDF_CODEPAGE_WIN_BALTIC	1257	Windows code pages
IG_PDF_CODEPAGE_MAC_CENTRAL_EUROPEAN	-9994	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_CROATIAN	-9993	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_ROMANIAN	-9992	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_CYRILLIC	-9991	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_UKRAINIAN	-9990	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_GREEK	-9989	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_TURKISH	-9988	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_HEBREW	-9987	Macintosh pseudo code pages
IG_PDF_CODEPAGE_MAC_ARABIC	-9986	Macintosh pseudo code pages

enumIGPDEPathElementType

Group: Enumeration

Description:

Constant values that describe path segment operators in PDE path elements.

Values:

IG_PDE_MOVE_TO	0	Designates m (moveto) operator, which moves the current point.
IG_PDE_LINE_TO	1	Designates l (lineto) operator, which appends a straight line segment from the current point.
IG_PDE_CURVE_TO	2	Designates c (curveto) operator, which appends a Bézier curve to the path.
IG_PDE_CURVE_TO_V	3	Designates v (curveto) operator, which appends a Bézier curve to the current path when the first control point coincides with initial point on the curve.
IG_PDE_CURVE_TO_Y	4	Designates y (curveto) operator, which appends a Bézier curve to the current path when the second control point coincides with final point on the curve.
IG_PDE_RECT	5	Designates re operator, which adds a rectangle to the current path.
IG_PDE_CLOSE_PATH	6	Designates h (closepath) operator, which closes the current subpath.

enumIGPDEPathOpFlags

Group: Enumeration

Description:

Flags for paint operators in a PDE path.

Values:

IG_PDE_INVISIBLE	0x00	Path is neither stroked nor filled, so it is invisible.
IG_PDE_STROKE	0x01	Stroke the path, as with the S (stroke) operator.
IG_PDE_FILL	0x02	Fills the path, using the nonzero winding number rule to determine the region to fill, as with the f (fill) operator.
IG_PDE_EO_FILL	0x04	Fills the path, using the even-odd rule to determine the region to fill, as with the f* (eofill) operator.

enumIGPDFBasicType

Group: Enumeration

Description:

Basic PDF objects.

Values:

IG_PDF_BASIC_NULL	0	Null object
IG_PDF_BASIC_INT	1	Integer object
IG_PDF_BASIC_FIXED	2	Fixed (real) object
IG_PDF_BASIC_BOOL	3	Boolean object
IG_PDF_BASIC_NAME	4	Name object
IG_PDF_BASIC_STRING	5	String object
IG_PDF_BASIC_DICT	6	Dictionary object
IG_PDF_BASIC_ARRAY	7	Array object
IG_PDF_BASIC_STREAM	8	Stream object

enumIGPDFPermsFlags

Group: Enumeration

Description:

Flags that describe permissions wanted and granted for a document. Not all permissions will be granted, if the document is protected or if the document is newer version than the application knows about.

Values:

IG_PDF_PERM_OPEN	0x01	The user is permitted to open and decrypt the document.
IG_PDF_PERM_SECURE	0x02	The user is permitted to change the document's security settings.
IG_PDF_PERM_PRINT	0x04	The user is permitted to print the document. Page Setup access is unaffected by this permission, since that affects Acrobat's preferences - not the document's. In the Document Security dialog, this corresponds to the Printing entry.
IG_PDF_PERM_EDIT	0x08	The user is permitted to edit the document more than adding or modifying text notes (see also IG_PDF_PERM_EDIT_NOTES). In the Document Security dialog, this corresponds to the Changing the Document entry.
IG_PDF_PERM_COPY	0x10	The user is permitted to copy information from the document to the clipboard. In the Document Security dialog, this corresponds to the Content Copying or Extraction entry.
IG_PDF_PERM_EDIT_NOTES	0x20	The user is permitted to add, modify, and delete text notes (see also IG_PDF_PERM_EDIT). In the Document Security dialog, this corresponds to the Authoring Comments and Form Fields entry.
IG_PDF_PERM_SAVE_AS	0x40	The user is permitted to perform a "Save As..." If both IG_PDF_PERM_EDIT and IG_PDF_PERM_EDIT_NOTES are disallowed, "Save" will be disabled but "Save As..." is enabled. The "Save As..." menu item is not necessarily disabled even if the user is not permitted to perform a "Save As...". **NOTE:** Not settable by clients.
IG_PDF_PERM_EXT	0x80	
IG_PDF_PRIV_PERM_FILL_AND_SIGN	0x100	Override other enumIGPDFPermsFlags bits. It allows user to fill-in or sign existing form or signature fields.
IG_PDF_PRIV_PERM_ACCESSIBLE	0x200	Override IG_PDF_PERM_COPY to enable Accessibility API. If a document is saved in Rev2 format (Acrobat 4.0 compatible), only IG_PDF_PERM_COPY bit is checked to determine Accessibility API state.

IG_PDF_PRIV_PERM_DOC_ASSEMBLY	0x400	Override various IG_PDF_PERM_EDIT bit and allow the following operations; page insert/delete/rotate and create bookmark and thumbnail.
IG_PDF_PRIV_PERM_HIGH_PRINT	0x800	This bit is supplement to IG_PDF_PERM_PRINT. If it is clear (disabled) only low quality printing (Print As Image) is allowed. Under UNIX platforms, where "Print As Image" doesn't exist, printing is disabled.
IG_PDF_PERM_OWNER	0x8000	The user is permitted to perform all operations, regardless of the permissions specified by the document. Unless this permission is set, the document's permissions will be reset to those in the document after a full save.
IG_PDF_PRIV_PERM_FORM_SUBMIT	0x10000	Should be set if user can submit forms outside of the browser. This bit is supplement to IG_PDF_PRIV_PERM_FILL_AND_SIGN.
IG_PDF_PRIV_PERM_FORM_SPAWN_TEMPL	0x20000	Should be set if user can spawn template pages. This bit will allow page template spawning even if IG_PDF_PERM_EDIT and IG_PDF_PERM_EDIT_NOTES are clear.
IG_PDF_PERM_ALL	0xFFFFFFFF	
IG_PDF_PERM_SETTABLE	IG_PDF_PERM_PRINT + IG_PDF_PERM_EDIT + IG_PDF_PERM_COPY + IG_PDF_PERM_EDIT_NOTES	The OR of all operations that can be set by the user in the Standard Security dialog (IG_PDF_PERM_PRINT + IG_PDF_PERM_EDIT + IG_PDF_PERM_COPY + IG_PDF_PERM_EDIT_NOTES)
IG_PDF_PERM_USER	IG_PDF_PERM_ALL - IG_PDF_PERM_OPEN - IG_PDF_PERM_SECURE	All permissions.

enumIGPDFRevision

Group: Enumeration

Description:

Specifies /R revision value.

Values:

IG_PDF_REVISION_2	2	Support by Acrobat 3.0 and up.
IG_PDF_REVISION_3	3	Support by Acrobat 5.0 and up.
IG_PDF_REVISION_4	4	Support by Acrobat 6.0 and up.

enumIGPDFSecurityInfoFlags

Group: Enumeration

Description:

Flags used to specify various information about the Acrobat viewer's security and permissions.

Values:

IG_PDF_INFO_HAS_USER_PW	IG_PDF_PERM_OPEN	The document has a user password.
IG_PDF_INFO_HAS_OWNER_PW	IG_PDF_PERM_SECURE	The document has an owner password.
IG_PDF_INFO_CAN_PRINT	IG_PDF_PERM_PRINT	The document can be printed.
IG_PDF_INFO_CAN_EDIT	IG_PDF_PERM_EDIT	The document can be modified, for example by adding notes, links, or bookmarks.
IG_PDF_INFO_CAN_COPY	IG_PDF_PERM_COPY	The document text and graphics can be copied to the clipboard.

enumIGPDFStdSecurityMethod

Group: Enumeration

Description:

Specifies standard security algorithms.

Values:

IG_PDF_STD_SECURITY_METHOD_RC4_V2	2	RC4 algorithm for encryption.
IG_PDF_STD_SECURITY_METHOD_AES_V1	5	AES algorithm for encryption with a zero initialized iv.
IG_PDF_STD_SECURITY_METHOD_AES_V2	6	AES algorithm for encryption with a random initialized iv.

GS PDF Component API

ImageGear GS PDF Component allows you to extract text from Portable Data Format (PDF) or PostScript documents. This ability is useful for finding certain words in the documents, for retrieving information, and for recovering text from Postscript or PDF files when the source is lost.

ImageGear can read the original PDF file and extract the textual content at just about 100% accuracy (English text is approximated at close to 100%, but some symbols in the extracted text, such as “-”, may be approximated incorrectly). This is not OCR output; rather the software scans the original source document (a PDF file), ignores all the format characters and notation and outputs all the words.

The text extraction routine **`IG_PDF_text_extract()`** extracts words in the same sequence as they are rendered by the document. This usually, but not always, follows the order that a human would read the words on a page. Within this sequence, words are separated by either space or new line depending on whether or not they fall on the same line. Each page is terminated with a formfeed.

IG_PDF_text_extract

Group: GS PDF Function

You can call PDF component text extracting function using two ways. Firstly, you may call this function through the macro defined in **`i_PDF.h`** public header file:

```
#define IG_PDF_text_extract(_lpszFileName, _lpszTextName,
_nPage, _nPageCount)((AT_ERRCOUNT (CACCUAPI *)( LPCHAR,LPSTR,
LPSTR, UINT, UINT))
IG_comm_function_call)("PDF.IG_PDF_text_extract",
(_lpszFileName), (_lpszTextName), (_nPage), (_nPageCount))
```

In this case your application will search all included public headers for this macro and call this function through **`IG_comm_function_call()`** component manager function call that is also determined in **`i_PDF.h`**. So if you are going to use PDF text extracting function in a multiple loop we would recommend you the second way of using this function type declaration that is also determined in **`i_PDF.h`** file:

```
typedef AT_ERRCOUNT (LPACCUAPI LPAFT_PDF_TEXT_EXTRACT)(
LPCHAR lpFileName,
LPCHAR lpTextFileName,
UINT nStartPage,
UINT nPageCount
);
```

So you should declare the variable of this function type and then use **`IG_comm_entry_request()`** to initialize this variable with correct value and call it then.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_text_extract(
LPSTR lpszFileName,
LPSTR lpszTextName,
UINT nStartPage,
UINT nPageCount);
```

Arguments:

lpFileName	Name of the PDF or PS document
lpTextFileName	Name of the output TXT file
nStartPage	Number of the first page
nPageCount	Total number of pages to be processed (starting at `nStartPage`)

Description:

This function extracts text from pages determined by arguments nStartPage and `nPageCount`. It extracts text from `nStartPage` to (`nStartPage + nPageCount-1`). It is necessary to include the **`"i_PDF.h"`** public header file, which provides the declaration of this function.

Return Value:

Error count

Example:

```
#include "i_PDF.h"
IG_PDF_text_extract( lpszMyDocument, lpszOutputText, 1, 10)
```

Index

Numerics

16-bit gray-scale Images
  Adjusting contrast 1274
  Adjusting contrast automatically 1276
  Checking 1283
  Converting to 8 bit 1280
  Downshifting 1286
  Loading and Displaying 205
  Pseudocoloring 1272
  Reducing to 8 bit 1286
16x8 LUT 211
1-bit images. 79
24-bit images. 78
4-bit images. 79
8-bit Images 79
  Promoting to 16-bit image 1285

A

AccuSoft Image Guarantee. 477
AIX 53
Alpha channel 211
  blending two images with alpha channels 875
AM_TID_META_BOOL 495
AM_TID_META_DOUBLE 497
AM_TID_META_FLOAT 496
AM_TID_META_INT16 494
AM_TID_META_INT32 494
AM_TID_META_INT64 495
AM_TID_META_INT8 493
AM_TID_META_RATIONAL_INT32. 496
AM_TID_META_RATIONAL_UNT32 496
AM_TID_META_STRING 497
AM_TID_META_UINT16 494
AM_TID_META_UINT32 495
AM_TID_META_UINT64 495
AM_TID_META_UINT8 494
AM_TID_RAW_DATA. 497
ANSI C 279
Anti-aliasing 108
anti-aliasing. 268, 273
Application-specific Data
  Saving 1151, 1167, 1169
ARGB 226
ART 1196
ART (.ART) files
  Loading into a HIGEAR image 1147, 1149, 1150
  Saving marks to 1148
ART Component
  named blocks 195
ART component
  versions 1195
ART Component Callback Functions
  using 195
ART Control Settings
  Modifying 1133
ART mark rotating
  getting angle 1193
  setting angle 1194
ART marks
  FREEHAND LINE 201
  HOLLOW ELLIPSE 201
ART_callback_register() 1131, 1197, 1199, 1200, 1202, ... 1204, 1206, 1208, 1209
ART_control_option_get() 1132
ART_control_option_set() 1130, 1133, 1148
ART_edit_paste() 1138
ART_edit_paste_count(). 1140
ART_edit_undo() 194, 1140, 1142, 1144
ART_edit_undo_clear() 1141
ART_edit_undo_record() 193, 194, 1143
ART_enable_rotation_get() 1145
ART_enable_rotation_set() 1145
ART_file_append(). 1146
ART_file_export() 194, 1147
ART_file_import() 195, 1148
ART_file_import_ex() 1150
ART_global_block_create() 1151, 1153
ART_global_block_delete() 1152
ART_global_block_first() 194, 1153, 1155
ART_global_block_next() 194, 1154
ART_global_block_query() . . . 1154, 1155, 1156
ART_group_access() 1157
ART_group_bring_to_front() 1158, 1175
ART_group_delete(). 1159
ART_group_edit_copy(). 1160
ART_group_edit_cut() 1161
ART_group_select() 1162
ART_group_send_to_back() 1163
ART_group_visible() 1163
ART_GUI_mark_paint() 194
ART_image_burn_in() 194, 1131, 1148

ART_MARK_ATTRIBUTES structure 195
ART_mark_block_create() 191, 192, 1166
ART_mark_block_create_ex() 192, 1168
ART_mark_block_delete() 1170
ART_mark_block_first() 1171, 1173
ART_mark_block_next() 1171, 1172
ART_mark_block_query() 1171, 1173, 1174
ART_mark_bring_to_front() 1175, 1185
ART_mark_count() 1176
ART_mark_create() 191, 192, 1177
ART_mark_delete() 192, 1178
ART_mark_first() 194, 1173, 1179, 1184
ART_mark_is_selected() 1181
ART_mark_is_visible() 1182
ART_mark_modify() 192, 1131, 1177, 1182
ART_mark_next() 194, 1184
ART_mark_query() 1131, 1185
ART_mark_rotate() 1186
ART_mark_select() 1187
ART_mark_select_count() 1188
ART_mark_selected_first() 193, 1189
ART_mark_selected_next() 1190
ART_mark_send_to_back() 1190
ART_mark_visible() 1191
ART_rotation_angle_get() 1193
ART_rotation_angle_set() 1193
ART_UNIX_LEGACY_FONTS_LOAD_GET . . 1195
ART_UNIX_LEGACY_FONTS_LOAD_SET 1196
ART_version_numbers() 1194
ASCII 85 encoding 253, 261, 275
ASCII file
  page width 598
ASCII HEX encoding 253, 261, 275
AT_COLORSPACE_CONVERSION_OPTIONS structure 147
AT_DIB 89
AT_MED_DCM_DISPLAY_SETTINGS . . 1300
AT_MED_MODALITY_RESCALE 1301
AT_MED_PIXEL_PADDING_SETTINGS 1302
AT_MED_VOI_WINDOW 1302
AT_MODE 909
AT_NR_ROI_MASK 928, 929
AT_PDE_COLORDATA 1640
AT_PDE_COLORDATA_CALGRAY 1635
AT_PDE_COLORDATA_CALRGB 1636
AT_PDE_COLORDATA_DEVICEN 1639
AT_PDE_COLORDATA_ICCBASED 1637
AT_PDE_COLORDATA_INDEXED 1638
AT_PDE_COLORDATA_LAB 1637
AT_PDE_COLORDATA_SEPARATION . 1639
AT_PDE_COLORRANGE 1634
AT_PDE_COLORSPEC 1627
AT_PDE_COLORVALUE 1626
AT_PDE_CONTENTATTRS 1644
AT_PDE_DASH 1627
AT_PDE_FILTERARRAY 1643
AT_PDE_FILTERSPEC 1643
AT_PDE_FONT_CREATEFROMSYSFONTPARAMS 1633
AT_PDE_FONT_INFO 1632
AT_PDE_FONTATTRS 1624
AT_PDE_GRAPHICSTATE 1628
AT_PDE_IMAGEATTRS 1642
AT_PDE_PSATTRS 1641
AT_PDE_TEXTSTATE 1629
AT_PDE_XYZCOLOR 1635
AT_PDF_BOOL 1608
AT_PDF_FIXED 1608
AT_PDF_FIXEDMATRIX 1610
AT_PDF_FIXEDPOINT 1609
AT_PDF_FIXEDQUAD 1611
AT_PDF_FIXEDRECT 1611
AT_PDF_FONT_METRICS 1613
AT_PDF_FONT_STYLES 1612
AT_PDF_PRINTPARAMS 1615
AT_PDF_SYSFONT_PLATDATA 1631
AT_POINT 90
AT_RECT 90, 599, 601, 795, 800, 809, 874, 877, 899, ... 903, 904
AT_RGB 91
AT_RGBQUAD 91
AT_STITCH 176
AUTO_POSITION 254

B

bitmap data 82
  1-bit images 82
  24-bit images 82
  4-bit images 82
  8-bit images 82
bitmap image 81
Bits Allocated 224, 1214, 1215
Bits Stored 224, 1214, 1215
Blending and Combining Images 122
Blocks (ART)
  Global Named Blocks 1167, 1169
  Named Blocks 1167, 1169
Burning marks into an image 1165
Byte order 1217
  Big Endian 1217
  Little Endian 1217
Bytes swapping 1287

C

C API 1312
C++ compiler 279
Callback Functions 164
After a mark has been created (See also LPFNIG_ART_CREATENOTIFYFUNC) 1199
After a mark has been deleted (See also LPFNIG_ART_DELETENOFIFYFUNC) 1200
For letting the developer draw the mark -- not the library (See also LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC) 1206
how to register 165
memory allocation (>1024) 1008
Registering 1131
When a mark attribute has been modified (See also LPFNIG_ART_MODIFYNOTIFYFUNC) 1204
When a mark attribute is about to be modified (See also LPFNIG_ART_SELECTNOTIFYFUNC) 1207
When ART encounters a mark it does not know how to draw (See also LPFNIG_ART_MODIFYNOTIFYFUNC) 1202
Canon 471
CIFF Non-Image Data Structure. 493
Clipboard 192, 604
copy 599
Copying array of marks 1136, 1160
copying images 128
cutting 601
Cutting array of marks 1137, 1161
Getting the current setting for paste 607
image dimensions 602
paste 603
pasting images from 129
Pasting marks into image 1139
Querying 1140
Region of interests(ROI) 604
CMYK 226
images 115
loading 94
support 45
working with 621
Code String 226, 1222
coefficients 302
color palette 79, 81
Color profile
CMYK 145
getting info about 631
getting info about global profile 634
Groups 144
Export Color Profile 145
Import Color Profile 144
Working Color Profile 145
Manager 143
API 145
new profile setting 632
resetting 636
RGB 145
setting new value for global profile 635
Color promotion 116
Color reduction 115
Color space
color space level getting 621
color space level setting 622
Columns 224, 1215
Compilation Flags. 280
Compiling 279
Component Manager API 187
component mixing 301
Components 49
Compressing 284
Compression 44
lossless 303
Constants
ART_BLOCK_ 1151, 1174
ART_BURN_IN_ 1166
ART_INVALID_ID 194
IG_ARITH_ 871
IG_COLOR_COMP_ 915
IG_COLOR_SPACE_ 878
IG_COMPASS_ 799
IG_COMPRESSION_ 864, 865, 867
IG_DIB_AREA_ 636, 639
IG_DRAW_FRAME_ 909
IG_EDGE_OP_ 910
IG_ENCRYPT_METHOD_ 911
IG_FILL_MSB 963, 965
IG_FORMAT_ 864, 865, 867
IG_INTERPOLATION_ 932, 934, 952
IG_LOAD_COLOR_ 965
IG_ROTATE_ 948
IG_ROTATE_CLIP 943, 945, 946
IG_SAVE_ 789, 1068
IG_TAG_TYPE_ 1125
MED_DCM_TS_ 220
Contrast Alteration 118

Control Parameters Reference 244, 266
Conversion between PDF and PostScript formats 834
Current Data Element 218, 219, 228
Data Field
Converting Data to a String 1263
Deleting 1221
Getting and Setting Data 229
Getting Index of 1219
Getting Info from 1216, 1217, 1220
Hierarchy 1220, 1228, 1229, 1231, 1233, 1235, 1236, 1238, 1242
Index 1236
Levels
Ascending 1228
Descending 1229
Moving to the First DE 1235
Moving to the Last DE 1238
Moving to the Next DE 1240
Moving to the Previous DE 1242
Moving to the Specified Index 1236

D

Data Callback 167
Data Elements 218
Deleting 228
Getting a Number of DEs in a Data Set 1224
Group Lengths 1243
Inserting new 225, 1222
Reading from 217
Writing to 217
Data Set
Creating 1215
CreatingSee MED_DCM_DS_create() 1215
Determining if empty 1225
Getting and Setting 224
Getting Info about 1224
Getting the number of DEs in a Data Set 224
Hierarchy 217, 219, 1228, 1230, 1231, 1233, 1235, 1236, 1238, 1240, 1242
Levels 219
The components of 218
Data Set Levels 219
Data Structures 89
Data Types 88
DataDict.h 226, 231, 1220, 1228, 1229, 1235, 1236, 1238, ... 1240, 1251, 1252
DCM.h 1215, 1222, 1228, 1229, 1231, 1233, 1235, 1236, 1238, 1240, 1245, 1257, 1262, 1264, 1267
DCM_CONTROL_LOAD_DETECT_SKIP_DIMSE 210
DCM_CONTROL_LOAD_SYNTAX 211
DCRAW Non-Image Data Structure 493
decomposition levels 302
Deployment license 67
DEPTH 244, 254, 273
deskew (de-skew) 907
Development license 66
Device Context 83
Device-Dependent Bitmap (DDB) 81, 83
Device-Independent Bitmap (DIB) 81
DICOM Data Set
deleting 1223
Existing 1223
DICOM format 50, 205
DICOM Image
Loading 1257
File Descriptor handle 1258
Preamble
Getting 1249
Setting 1249
DIMSE commands
Allowing or Ignoring 210
display depth 267, 268, 273
display driver 267
display resolution
horizontal 272
vertical 272
Dithering 108
DPI
convert to PPM 628
Drawing
Using FreeHand Draw Annotation Object 201
dyadic decomposition 302
dynamic loader 280

E

Edits
Undoing 193
Element Number
Getting from a Tag 1251
Embedded marks
TIFF file 190
Encoding 284
encoding scheme 253, 261, 275
Entropy Coding 301
enumIGPDEContentFlags 1659
enumIGPDEContentGetResourceFlags 1659
enumIGPDEElementCopyFlags 1658
enumIGPDEFontCreateFlags 1652
enumIGPDEFontCreateNeedFlags 1653
enumIGPDEFontProtection 1650

enumIGPDEGraphicStateWasSetFlags 1655
enumIGPDEImageAttrFlags. 1657
enumIGPDEImageDataFlags 1656
enumIGPDEInsertElement 1651
enumIGPDEPathElementType 1662
enumIGPDEPathOpFlags 1663
enumIGPDEPSAttrFlags 1658
enumIGPDESoftMaskCreateFlags 1657
enumIGPDETextFlags 1650
enumIGPDEType 1648
enumIGPDEXGroupCreateFlags 1657
enumIGPDFBasicType 1664
enumIGPDFBookmarkFlags 1645
enumIGPDFCharset 1646
enumIGPDFCodePages 1661
enumIGPDFDestinationType 1646
enumIGPDFFixedValues 1646
enumIGPDFFontFlags 1652
enumIGPDFPageNumber 1651
enumIGPDFSysFontMatchFlags 1654
enumIGPDFSysFontPackageType 1654
enumIGPDFWordFinderVersion 1660
enumIGPDFWordFlags 1660
error handing. 162
Evaluation license 66
Excluding Files Licensed Only for Your Own Use 281
EXIF Makernote 520
EXIF-JPEG Non-image Data Structure 493
EXIF-TIFF Non-Image Data Structure. 493
expose event 92

F

File Formats
- AVI 434
- BMP 322
- BTR 325
- converting 100
- de facto standards 285
- EPS 51
- ICO 383
- IFF 385
- IMT 389
- JPEG-JFIF 391
- KFX 405
- LV 406
- MAC 406
- MPEG 412
- PBM 416
- PCD 418
- PCX 419
- PDF 51
- PGM 421
- PNG 423
- PNM 426
- PPM 428
- proprietary formats 285
- PS 51
- PSD 316
- PTOCA 430
- RAS 432
- Scitex CT 436
- SGI 439
- standards of storage for graphics data 285
- TXT 458

file formats
- EPS 264
- PDF 263
- PS 264

FILE_NAME 267
Filter Flags
- IG_FLTR_DETECTSUPPORT 131
- IG_FLTR_MPAGEREADPSUPPORT 131
- IG_FLTR_MPAGEWRITEPSUPPORT 131
- IG_FLTR_MPDATASUPPORT 131
- IG_FLTR_PAGEDELETESUPPORT 131
- IG_FLTR_PAGEINSERTSUPPORT 131
- IG_FLTR_PAGEREADSUPPORT 131
- IG_FLTR_PAGESWAPSUPPORT 131

Filtering modes 302
- convolution-based 302
- lifting-based 302

Filters
- File Format
  - detecting 132
- Filter Control 135
- Getting Information and Sorting Images 133
- Image Loading 132
- Image Saving 134
- metadata callback functions 140
- Non-Image data format 138
- Non-Image Data Processing 137

Format conversion 43
Formats supported in ImageGear 46
Fujifilm 473
Functions
- Saving Functions 206

G

Gamma correction 1274, 1279
gamma correction 112
Geometric Layout 102

Ghostscript 58
GIF Non-image Data Structue 493
GIF Non-image Data Structure 549
Global Blocks 191
Global Control Parameters 146
Global Named Blocks
    Creating 1151
    Deleting 1153
    Getting the first block 1154
    Getting the next block 1155
    Iterating through 1155
    iterating through 194
    Querying data of 1156
    Querying size of data 1156
GRAPHICS_ALPHA 268
Group Lengths 1243
    Saving with image 213
Group Number 1233
    Getting from a Tag 1252
Groups of marks
    Bringing to the front 1158
    Copying to the clipboard 1160
    Cutting to clipboard 1161
    Deleting 1159
    Hiding or showing 1164
    Selecting or deselecting 1162
    Sending to the back 1163
    Setting access of 1157
    Setting Group names 1167, 1169

H

HD Photo 371
HDS 1246
HDS (Internal Data Set) 220, 1247
HDS Structure 1214
HDS table 1218
HEIGHT 254
Help Files 36
Hierarchy of a Data Set 217, 219
    returning current level in 1240
HIG_PDE_CLIP 1514
HIG_PDE_COLORSPACE 1480
HIG_PDE_CONTAINER 1520
HIG_PDE_CONTENT 1475
HIG_PDE_ELEMENT 1504
HIG_PDE_FONT 1486
HIG_PDE_FORM 1525
HIG_PDE_GROUP 1530
HIG_PDE_IMAGE 1532
HIG_PDE_OBJECT 1474
HIG_PDE_PATH 1548
HIG_PDE_PLACE 1553
HIG_PDE_POSTSCRIPT 1557
HIG_PDE_SHADING 1567
HIG_PDE_SOFTMASK 1562
HIG_PDE_TEXT 1568
HIG_PDE_TEXTITEM 1593
HIG_PDE_XGROUP 1602
HIG_PDE_XOBJECT 1607
HIG_PDF_ACTION 1358
HIG_PDF_ATOM 1364
HIG_PDF_BASARR 1342
HIG_PDF_BASBOOL 1332
HIG_PDF_BASDICT 1349
HIG_PDF_BASFIXED 1336
HIG_PDF_BASINT 1334
HIG_PDF_BASNAME 1340
HIG_PDF_BASNULL 1356
HIG_PDF_BASOBJ 1330
HIG_PDF_BASSTR 1338
HIG_PDF_BOOKMARK 1366
HIG_PDF_DESTINATION 1389
HIG_PDF_DOC 1399
HIG_PDF_FONT 1417
HIG_PDF_LAYER 1429
HIG_PDF_PAGE 1441
HIG_PDF_STREAM 1384
HIG_PDF_STYLE 1455
HIG_PDF_SYSENCODING 1466
HIG_PDF_SYSFONT 1458
HIG_PDF_WORD 1449
HIG_PDF_WORDFINDER 1446
HIGEAR
    Association with DICOM Data 216
High Bit 224, 1214, 1215, 1285
    Changing 1281
    Setting the Value of 1285
Histogram
    Clearing the Histogram 1281
    Creating the Histogram for an 8g or 16g image 1282
HP-UX 53
HSV 226
Hue
    Deriving a Hue value from an RGB value 942

I

i_ART.h file 195
ICC
    export devices 144
    import devices 143
    working devices 144

ICC callback 1099
ICC input profiles 269
ICC profile 1099
ICC_PROFILE_MODE 204
IG Components
ART 50
GIF/TIFF-LZW 50
MD (Medical) 50
PDF 51
IG Functions
Platform-specific functions 92
IG_close 92
IG_initialize 92
IG_ascii_import() 597
IG_ascii_page_width_get() 598
IG_clipboard_copy() 599
IG_clipboard_cut() 600
IG_clipboard_dimensions() 602
IG_clipboard_paste() 603
IG_clipboard_paste_available() 602, 603
IG_clipboard_paste_available_ex() 604
IG_clipboard_paste_merge() 608
IG_clipboard_paste_merge_ex() 605
IG_clipboard_paste_op_get() 607
IG_close 69, 92, 609
IG_COLOR_SPACE_ constants 621
IG_color_space_level_get() 95, 621
IG_color_space_level_set() 95, 622
IG_colorspace_conversion_apply_to_image . 609
IG_colorspace_conversion_apply_to_raster. . 610
IG_colorspace_conversion_create 612
IG_colorspace_conversion_create_from_dib_info 614
IG_colorspace_conversion_create_from_image 615
IG_colorspace_conversion_destroy 616
IG_colorspace_conversion_is_applicable . . . 617
IG_colorspace_conversion_result_channels_build 618
IG_colorspace_conversion_result_palette_get 620
IG_comm_comp_attach() 187, 237, 265, 623
IG_comm_comp_check() 187, 237, 265, 624
IG_comm_comp_list() 239, 265, 625
IG_comm_entry_request() 188, 626
IG_comm_function_call() 187, 627
IG_COMPRESSION_CCITT_G3 . . . . . 244, 266
IG_COMPRESSION_CCITT_G32D . . . 244, 266
IG_COMPRESSION_CCITT_G4 . . . . . 244, 266
IG_COMPRESSION_DEFLATE 244
IG_COMPRESSION_JPEG 244, 266
IG_COMPRESSION_LZW 244
IG_COMPRESSION_NONE 244, 266
IG_COMPRESSION_RLE 244
IG_convert_DPI_to_PPM() 628
IG_convert_PPM_to_DPI() 628, 629
IG_cpm_image_embedded_profile_check . . . 629
IG_cpm_image_profile_get 630
IG_cpm_image_profile_get() 146, 630
IG_cpm_image_profile_set() 146, 632
IG_cpm_profile_get() 146, 633
IG_cpm_profile_set() 146, 634
IG_cpm_profiles_reset() 146, 636
IG_DIB_area_get() 636
IG_DIB_area_size_get() 640
IG_DIB_bit_depth_get 642
IG_DIB_channel_count_get 642
IG_DIB_channel_depth_get 643
IG_DIB_channel_depths_get 644
IG_DIB_colorspace_get 645
IG_DIB_column_get() 646
IG_DIB_column_set() 647
IG_DIB_height_get 649
IG_DIB_info_copy 649
IG_DIB_info_create 650
IG_DIB_info_delete 651
IG_DIB_info_raster_size_get 652
IG_DIB_legacy_bit_depth_get 653
IG_DIB_line_get() 654
IG_DIB_line_set() 655
IG_DIB_palette_alloc 657
IG_DIB_palette_length_get 658
IG_DIB_palette_pointer_get 659
IG_DIB_palette_size_get 660
IG_DIB_pix_get 664
IG_DIB_pix_set 665
IG_DIB_pixel_array_size_get 660
IG_DIB_pixel_get() 661
IG_DIB_pixel_set() 662
IG_DIB_raster_get() 666
IG_DIB_raster_set() 668
IG_DIB_raster_size_get() 669
IG_DIB_resolution_get 669
IG_DIB_resolution_set 670
IG_DIB_row_get() 672
IG_DIB_row_set() 674
IG_DIB_width_get 675
IG_display_animation_delay_get() 676
IG_display_animation_delay_set() 676
IG_display_clip_rect_get 92
IG_display_clip_rect_set 92
IG_display_contrast() 1276
IG_display_image() 1272, 1273, 1275
IG_dspl_antialias_get() 677
IG_dspl_antialias_get_ex() 678
IG_dspl_antialias_set_ex 680

IG_dspl_artx_image_draw 682
IG_dspl_background_get() 683
IG_dspl_background_set() 684
IG_dspl_DDB_create() 685
IG_dspl_DDB_import() 687
IG_dspl_device_to_image() 688
IG_dspl_device_to_image_d() 690
IG_dspl_dithering_get() 691
IG_dspl_dithering_set() 691
IG_dspl_foreground_get() 692
IG_dspl_foreground_set() 693
IG_dspl_free_grp_id_get() 694
IG_dspl_gamma_correction_LUT_build() . . . 695
IG_dspl_gamma_correction_set() 696
IG_dspl_grayscale_LUT_exists 698
IG_dspl_grayscale_LUT_update_from . 698, 699
IG_dspl_image_wipe() 706
IG_dspl_layout_get() 708
IG_dspl_layout_set 736
IG_dspl_layout_set() 95, 710
IG_dspl_LUT_get() 713
IG_dspl_LUT_set() 714
IG_dspl_orientation_get() 716
IG_dspl_orientation_set() 717
IG_dspl_palette_create() 718
IG_dspl_palette_get() 720
IG_dspl_palette_handle() 719
IG_dspl_palette_set() 721
IG_dspl_PPM_correct_get() 722
IG_dspl_PPM_correct_set() 723
IG_dspl_resize_handle() 724
IG_dspl_scroll_get() 726
IG_dspl_scroll_handle() 727
IG_dspl_scroll_set() 729
IG_dspl_scroll_to() 731
IG_dspl_scroll_to_ex() 732
IG_dspl_transparency_get() 733
IG_dspl_transparency_set() 734
IG_dspl_zoom_get() 738
IG_dspl_zoom_set() 739
IG_dspl_zoom_to_rect() 740
IG_err_callback_get() 741
IG_err_count_get() 744
IG_err_error_check() 744
IG_err_error_get() 745
IG_err_error_set() 747
IG_err_record_get() 748
IG_err_stack_clear() 750
IG_error_check() 750, 755
IG_error_clear() 751, 753
IG_error_get() 751, 752
IG_error_set() 753, 1105, 1107, 1112, 1113
IG_file_IO_register() 755, 1114
IG_fltr_compressionlist_get() 134, 135, 756
IG_fltr_ctrl_get() 136, 759
IG_fltr_ctrl_list() 136, 761
IG_fltr_ctrl_set() 136, 762
IG_fltr_detect_FD() 132, 763
IG_fltr_detect_file() 132, 764
IG_fltr_detect_mem() 132, 765
IG_FLTR_DETECTSUPPORT 243, 265
IG_fltr_formatlist_get() 133, 766
IG_fltr_formatlist_sort() 133, 768
IG_fltr_ICC_callback_get 769
IG_fltr_ICC_callback_set 770
IG_fltr_info_get() 133, 134, 771
IG_fltr_load_FD_format 774
IG_fltr_load_file() 132, 133, 140, 775
IG_fltr_load_file_format() 133, 776
IG_fltr_metad_callback_get() 142, 777
IG_fltr_metad_callback_set() 142, 778
IG_fltr_metad_update_file 780
IG_FLTR_MPAGEREADPSUPPORT . 243, 265
IG_FLTR_MPAGEWRITEPSUPPORT 243, 265
IG_fltr_pagecount_FD_format 782
IG_fltr_pagecount_file_format() 133, 783
IG_fltr_pagedelete_file() 134, 784
IG_FLTR_PAGEDELETESUPPORT . . 243, 265
IG_fltr_pageinfo_get() 134, 785
IG_FLTR_PAGEINSERTSUPPORT . . 243, 265
IG_FLTR_PAGEREADSUPPORT 243, 265
IG_fltr_pageswap_file() 134, 786
IG_FLTR_PAGESWAPSUPPORT 243, 265
IG_fltr_raster_plane_callback_get 787
IG_fltr_raster_plane_callback_set 788
IG_fltr_save_file() 134, 135, 140, 206, 789
IG_fltr_savelist_get() 134, 135, 790
IG_FX_blur() 794
IG_FX_chroma_key() 795
IG_FX_diffuse() 797
IG_FX_emboss() 798
IG_FX_motion() 800
IG_FX_noise() 801
IG_FX_pixelate 803
IG_FX_posterize() 805
IG_FX_spotlight() 806
IG_FX_stitch() 808
IG_FX_texture() 809
IG_FX_twist() 811
IG_FX_watermark() 812
IG_gctrl_item_by_index_get() 147, 813
IG_gctrl_item_count_get() 147, 814
IG_gctrl_item_get() 147, 815
IG_gctrl_item_id_get() 147

IG_gctrl_item_set() 145, 147, 816
IG_image_bits_per_channel_get 817
IG_image_blend_with_alpha 818
IG_image_burn_in() 1165
IG_image_channel_add 819
IG_image_channel_copy_create. 820
IG_image_channel_count_get 822
IG_image_channel_depths_change 824
IG_image_channel_remove 825
IG_image_channel_update 826
IG_image_channels_combine. 827
IG_image_channels_separate 828
IG_image_colorspace_convert 829
IG_image_colorspace_get 830
IG_image_compression_type_get() 831
IG_image_control_set() 1261
IG_image_convert() 832
IG_image_create. 834
IG_image_create_alpha() 835
IG_image_create_DIB_ex() 839
IG_image_create_empty. 841
IG_image_delete() 1257
IG_image_DIB_export 843
IG_image_DIB_export_size_calc. 845
IG_image_DIB_import. 847
IG_image_DIB_info_get 848
IG_image_DIB_palette_pntr_get(). 849
IG_image_DIB_raster_pntr_get 850
IG_image_dimensions_get(). 636, 851
IG_image_duplicate() 852
IG_image_grayscale_LUT_copy_get. 852
IG_image_grayscale_LUT_exists 853
IG_image_grayscale_LUT_update_from 854
IG_image_is_gray() 854
IG_image_is_signed_set() 855
IG_image_is_valid() 856
IG_image_orientation_get() 857
IG_image_orientation_set() 858
IG_image_resolution_get(). 859
IG_image_resolution_set() 861
IG_info_get() 95
IG_info_get_ex() 863
IG_info_get_FD_ex() 865
IG_info_get_mem_ex() 866
IG_initialize 69
IG_IP_alpha_create() 868
IG_IP_arithmetic() 870
IG_IP_arithmetic_rect() 871
IG_IP_blend_percent 872
IG_IP_blend_percent() 872
IG_IP_blend_with_alpha() 874
IG_IP_blend_with_LUT() 877
IG_IP_color_combine_ex. 879
IG_IP_color_combine_ex() 879
IG_IP_color_convert() 880
IG_IP_color_count_get. 880
IG_IP_color_promote() 881
IG_IP_color_reduce_bayer() 882
IG_IP_color_reduce_diffuse() 883
IG_IP_color_reduce_halftone() 884
IG_IP_color_reduce_median_cut() 885
IG_IP_color_reduce_octree() 886
IG_IP_color_reduce_popularity(). 888
IG_IP_color_reduce_to_bitonal(). 889
IG_IP_color_separate(). 890
IG_IP_contrast_adjust_ex. 893
IG_IP_contrast_equalize() 895
IG_IP_contrast_gamma() 896
IG_IP_contrast_invert() 898
IG_IP_contrast_stretch(). 900
IG_IP_convert_to_gray() 901
IG_IP_convolve_matrix(). 901
IG_IP_crop(). 904
IG_IP_decrypt() 905
IG_IP_deskew_angle_find(). 906
IG_IP_despeckle() 907
IG_IP_draw_frame(). 909
IG_IP_edge_map() 910
IG_IP_encrypt() 911
IG_IP_flip() 912
IG_IP_gaussian_blur() 913
IG_IP_histo_clear() 914
IG_IP_histo_tabulate() 914
IG_IP_median() 917
IG_IP_merge() 918
IG_IP_NR_ROI_control_get() 921
IG_IP_NR_ROI_mask_associate(). 924
IG_IP_NR_ROI_mask_delete() 925
IG_IP_NR_ROI_to_HIGEAR_mask(). 601
IG_IP_pseudocolor_limits() 929
IG_IP_pseudocolor_small_grads() 929, 931
IG_IP_resize(). 931
IG_IP_resize_bkgrnd() 933
IG_IP_resize_bkgrnd_ex() 936
IG_IP_resize_ex(). 939
IG_IP_RGB_to_hue(). 942
IG_IP_rotate_any_angle_bkgrnd(). 944
IG_IP_rotate_any_angle_ex(). 945
IG_IP_rotate_compute_size(). 946
IG_IP_rotate_multiple_90() 948
IG_IP_sharpen() 949
IG_IP_smooth() 950
IG_IP_swap_red_blue() 951
IG_IP_thumbnail_create() 952

IG_IP_thumbnail_create_ex() 953
IG_IP_transform_with_LUT() 955
IG_IP_unsharp_mask() 956
IG_lic_solution_name_set() 958
IG_load_alpha_mode_get 958
IG_load_alpha_mode_set 959
IG_load_auto_detect_get() 960
IG_load_auto_detect_set() 961
IG_load_CCITT_FD() 962
IG_load_color_reduction_set() 965
IG_load_FD() 968
IG_load_FD_CB() 970
IG_load_FD_CB_ex 971
IG_load_file() 132, 973, 1257, 1258
IG_load_file_display() 974
IG_load_mem() 975
IG_load_mem_CB() 977
IG_load_raw_FD() 979
IG_load_raw_file() 981
IG_load_raw_mem() 982
IG_load_size_get() 985
IG_load_size_set() 986
IG_load_tag_CB_register() 987, 1122
IG_load_thumbnail() 988
IG_load_tiles_stitch_mem() 995
IG_LUT_copy 997
IG_LUT_copy_to_byte_array 998
IG_LUT_copy_to_word_array 998
IG_LUT_create 999
IG_LUT_destroy 1000
IG_LUT_input_depth_get 1001
IG_LUT_input_is_signed_get 1001
IG_LUT_is_valid 1002
IG_LUT_item_get 1002
IG_LUT_item_set 1003
IG_LUT_length_get 1004
IG_LUT_output_depth_get 1004
IG_LUT_output_is_signed_get 1005
IG_LUT_size_get 1005
IG_LUT_update_from_byte_array 1006
IG_LUT_update_from_word_array 1007
IG_mem_CB_register() . 1007, 1109, 1110, 1111
IG_mpf_info_get() 1009
IG_mpf_page_count_get() 1010
IG_mpf_page_delete() 1011
IG_mpf_page_get() 1012
IG_mpf_page_info_get() 1013
IG_mpf_page_load() 1016
IG_mpf_page_save() 1017
IG_mpf_page_swap() 1019
IG_mpf_page_unload() 1020
IG_mpf_tile_count_get() 1021
IG_mpi_CB_get() 1022
IG_mpi_CB_reset() 1024
IG_mpi_CB_reset_all() 1025
IG_mpi_CB_set() 1027
IG_mpi_close() 1030
IG_mpi_create() 1031
IG_mpi_delete() 1032
IG_mpi_file_open() 1032
IG_mpi_info_get() 1035
IG_mpi_is_valid() 1037
IG_mpi_page_count_get() 1038
IG_mpi_page_count_set() 1039
IG_mpi_page_delete() 1041
IG_mpi_page_get() 1042
IG_mpi_page_is_valid() 1043
IG_mpi_page_set() 1043
IG_page_count_get_FD() 1046
IG_palette_get 1049
IG_palette_load 1050
IG_palette_set() 1053
IG_PDE_clip_add_element 1515
IG_PDE_clip_clone 1515
IG_PDE_clip_create 1514
IG_PDE_clip_enumerate_elements 1518
IG_PDE_clip_get_element 1516
IG_PDE_clip_get_element_count 1517
IG_PDE_clip_remove_elements 1517
IG_PDE_colorspace_create 1480
IG_PDE_colorspace_get_base_color_components 1482
IG_PDE_colorspace_get_base_name 1481
IG_PDE_colorspace_get_color_components 1484
IG_PDE_colorspace_get_ctable 1482
IG_PDE_colorspace_get_hival 1483
IG_PDE_colorspace_get_name 1484
IG_PDE_colorspace_release 1485
IG_PDE_container_create 1520
IG_PDE_container_get_content 1521
IG_PDE_container_get_dictionary 1522
IG_PDE_container_get_mctag 1523
IG_PDE_container_set_content 1521
IG_PDE_container_set_dictionary 1523
IG_PDE_container_set_mctag 1524
IG_PDE_content_add_element 1476
IG_PDE_content_create 1475
IG_PDE_content_get_attrs 1478
IG_PDE_content_get_default_color_space . 1478
IG_PDE_content_get_element 1476
IG_PDE_content_get_element_count 1477
IG_PDE_content_remove_element 1477
IG_PDE_element_clone 1505
IG_PDE_element_get_bbox 1507

IG_PDE_element_get_clip 1508
IG_PDE_element_get_dictionary 1511
IG_PDE_element_get_gstate 1509
IG_PDE_element_get_matrix 1510
IG_PDE_element_get_type 1504
IG_PDE_element_get_unique_id 1512
IG_PDE_element_has_gstate 1512
IG_PDE_element_is_at_point 1505
IG_PDE_element_is_at_rect 1506
IG_PDE_element_release 1513
IG_PDE_element_set_clip 1508
IG_PDE_element_set_gstate 1509
IG_PDE_element_set_matrix 1511
IG_PDE_font_create 1488
IG_PDE_font_create_from_sysfont 1490
IG_PDE_font_create_from_sysfont_and_encoding 1491
IG_PDE_font_create_from_sysfont_with_params 1491
IG_PDE_font_create_tounicode_now 1492
IG_PDE_font_create_widths_now 1493
IG_PDE_font_embed_now 1494
IG_PDE_font_embed_now_dont_subset . . . 1494
IG_PDE_font_get_attrs 1492
IG_PDE_font_get_codebyte_count 1495
IG_PDE_font_get_create_need_flags 1495
IG_PDE_font_get_onebyte_encoding 1496
IG_PDE_font_get_sysencoding 1497
IG_PDE_font_get_sysfont 1497
IG_PDE_font_get_widths 1498
IG_PDE_font_get_widths_now 1498
IG_PDE_font_is_embedded 1499
IG_PDE_font_is_multibyte 1499
IG_PDE_font_release 1503
IG_PDE_font_set_sysencoding 1500
IG_PDE_font_set_sysfont 1500
IG_PDE_font_subset_now 1501
IG_PDE_font_sum_widths 1501
IG_PDE_font_translate_glyphids_to_unicode 1502
IG_PDE_form_clone 1526
IG_PDE_form_create 1525
IG_PDE_form_get_content 1527
IG_PDE_form_get_xgroup 1528
IG_PDE_form_has_xgroup 1528
IG_PDE_form_set_content 1527
IG_PDE_form_set_xgroup 1529
IG_PDE_group_create 1530
IG_PDE_group_get_content 1531
IG_PDE_group_set_content 1531
IG_PDE_image_create 1533
IG_PDE_image_get_attrs 1535
IG_PDE_image_get_color_mask 1536
IG_PDE_image_get_colorspace 1537
IG_PDE_image_get_data 1537
IG_PDE_image_get_data_length 1538
IG_PDE_image_get_data_stream 1539
IG_PDE_image_get_decode_array 1539
IG_PDE_image_get_filter_array 1540
IG_PDE_image_get_matte_array 1541
IG_PDE_image_get_soft_mask 1541
IG_PDE_image_has_soft_mask 1542
IG_PDE_image_is_data_encoded 1535
IG_PDE_image_is_xobject 1542
IG_PDE_image_set_color_mask 1543
IG_PDE_image_set_colorspace 1544
IG_PDE_image_set_data 1544
IG_PDE_image_set_data_stream 1545
IG_PDE_image_set_decode_array 1546
IG_PDE_image_set_matte_array 1547
IG_PDE_image_set_soft_mask 1547
IG_PDE_object_get_type 1474
IG_PDE_path_add_segment 1549
IG_PDE_path_create 1548
IG_PDE_path_get_data 1550
IG_PDE_path_get_paint_op 1551
IG_PDE_path_set_data 1551
IG_PDE_path_set_paint_op 1552
IG_PDE_place_create 1553
IG_PDE_place_get_dictionary 1554
IG_PDE_place_get_mctag 1555
IG_PDE_place_set_dictionary 1555
IG_PDE_place_set_mctag 1556
IG_PDE_postscript_create 1557
IG_PDE_postscript_get_attrs 1558
IG_PDE_postscript_get_data 1559
IG_PDE_postscript_get_data_stream 1560
IG_PDE_postscript_set_data 1559
IG_PDE_postscript_set_data_stream 1560
IG_PDE_shading_create 1567
IG_PDE_softmask_create 1562
IG_PDE_softmask_create_from_name 1563
IG_PDE_softmask_get_backdrop_color . . . 1565
IG_PDE_softmask_get_form 1564
IG_PDE_softmask_get_name 1566
IG_PDE_softmask_set_backdrop_color 1565
IG_PDE_softmask_set_form 1564
IG_PDE_text_add 1570
IG_PDE_text_add_item 1571
IG_PDE_text_create 1570
IG_PDE_text_get_advance 1572
IG_PDE_text_get_bbox 1573
IG_PDE_text_get_byte_count 1578
IG_PDE_text_get_char_count 1578
IG_PDE_text_get_font 1574

IG_PDE_text_get_gstate 1575
IG_PDE_text_get_item. 1576
IG_PDE_text_get_matrix 1577
IG_PDE_text_get_quad 1579
IG_PDE_text_get_run_for_char. 1580
IG_PDE_text_get_runs_count 1579
IG_PDE_text_get_state 1581
IG_PDE_text_get_stroke_matrix 1582
IG_PDE_text_get_text 1582
IG_PDE_text_is_at_point. 1583
IG_PDE_text_is_at_rect 1584
IG_PDE_text_remove. 1585
IG_PDE_text_remove_items 1586
IG_PDE_text_replace_chars. 1587
IG_PDE_text_run_get_char_count. 1588
IG_PDE_text_run_get_char_offset 1587
IG_PDE_text_run_set_font 1588
IG_PDE_text_run_set_gstate 1589
IG_PDE_text_run_set_matrix 1590
IG_PDE_text_run_set_state 1590
IG_PDE_text_run_set_stroke_matrix. 1591
IG_PDE_text_split_run_at 1591
IG_PDE_textitem_copy_text 1594
IG_PDE_textitem_create 1593
IG_PDE_textitem_get_font 1595
IG_PDE_textitem_get_gstate 1596
IG_PDE_textitem_get_matrix 1598
IG_PDE_textitem_get_state 1599
IG_PDE_textitem_get_text_length. 1597
IG_PDE_textitem_remove_chars 1600
IG_PDE_textitem_replace_chars 1600
IG_PDE_textitem_replace_text 1601
IG_PDE_textitem_set_font. 1595
IG_PDE_textitem_set_gstate 1597
IG_PDE_textitem_set_matrix. 1598
IG_PDE_textitem_set_state 1599
IG_PDE_xgroup_create 1602
IG_PDE_xgroup_get_colorspace 1603
IG_PDE_xgroup_get_isolated 1604
IG_PDE_xgroup_get_knockout 1605
IG_PDE_xgroup_set_colorspace 1603
IG_PDE_xgroup_set_isolated 1604
IG_PDE_xgroup_set_knockout 1605
IG_PDE_xobject_create 1607
IG_PDF_action_create 1358
IG_PDF_action_create_destination 1359
IG_PDF_action_create_filename 1360
IG_PDF_action_delete 1360
IG_PDF_action_get_destination. 1361
IG_PDF_action_get_dictionary 1361
IG_PDF_action_get_filename 1362
IG_PDF_action_get_type 1363
IG_PDF_atom_from_string 1364
IG_PDF_atom_get_string. 1365
IG_PDF_basarr_create 1342
IG_PDF_basarr_get 1344
IG_PDF_basarr_get_length 1343
IG_PDF_basarr_put 1344
IG_PDF_basarr_put_bool. 1346
IG_PDF_basarr_put_fixed 1346
IG_PDF_basarr_put_int 1345
IG_PDF_basarr_put_name 1347
IG_PDF_basarr_remove 1348
IG_PDF_basarr_remove_nth 1348
IG_PDF_basbool_create. 1332
IG_PDF_basbool_get_value. 1333
IG_PDF_basdict_create 1349
IG_PDF_basdict_get. 1351
IG_PDF_basdict_known. 1350
IG_PDF_basdict_put 1351
IG_PDF_basdict_put_bool 1353
IG_PDF_basdict_put_fixed 1353
IG_PDF_basdict_put_int 1352
IG_PDF_basdict_put_name 1354
IG_PDF_basdict_remove 1355
IG_PDF_basfixed_create 1336
IG_PDF_basfixed_get_value 1337
IG_PDF_basint_create 1334
IG_PDF_basint_get_value 1335
IG_PDF_basname_create 1340
IG_PDF_basname_get_value 1341
IG_PDF_basnull_create 1356
IG_PDF_basobj_get_type. 1330
IG_PDF_basobj_release 1330
IG_PDF_basobj_remove 1331
IG_PDF_basstr_create 1338
IG_PDF_basstr_get_value 1339
IG_PDF_bookmark_add_child. 1368
IG_PDF_bookmark_add_new_child 1368
IG_PDF_bookmark_add_new_sibling 1369
IG_PDF_bookmark_add_next 1369
IG_PDF_bookmark_add_prev 1370
IG_PDF_bookmark_add_subtree 1371
IG_PDF_bookmark_delete. 1371
IG_PDF_bookmark_find_title 1372
IG_PDF_bookmark_get_action 1372
IG_PDF_bookmark_get_color 1373
IG_PDF_bookmark_get_count. 1373
IG_PDF_bookmark_get_first_child 1374
IG_PDF_bookmark_get_flags 1374
IG_PDF_bookmark_get_indent 1375
IG_PDF_bookmark_get_last_child 1375
IG_PDF_bookmark_get_next. 1376
IG_PDF_bookmark_get_parent 1376

IG_PDF_bookmark_get_prev. 1377
IG_PDF_bookmark_get_title 1378
IG_PDF_bookmark_has_children 1378
IG_PDF_bookmark_is_open 1379
IG_PDF_bookmark_remove. 1379
IG_PDF_bookmark_remove_action. 1380
IG_PDF_bookmark_set_action 1380
IG_PDF_bookmark_set_color 1381
IG_PDF_bookmark_set_flags 1381
IG_PDF_bookmark_set_open 1382
IG_PDF_bookmark_set_title 1382
IG_PDF_bookmark_unlink 1383
IG_PDF_destination_create 1389
IG_PDF_destination_delete 1390
IG_PDF_destination_get_explicit_attrs 1391
IG_PDF_destination_get_named_attrs. 1391
IG_PDF_destination_get_type 1392
IG_PDF_destination_remove 1393
IG_PDF_destination_resolve 1393
IG_PDF_dictionary_create. 1395
IG_PDF_dictionary_get_layer 1396
IG_PDF_dictionary_get_layer_count. 1397
IG_PDF_dictionary_get_unique_id 1397
IG_PDF_dictionary_get_vis_policy. 1398
IG_PDF_dictionary_release 1398
IG_PDF_doc_create 1401
IG_PDF_doc_create_new_page 1401
IG_PDF_doc_create_wordfinder 1409
IG_PDF_doc_create_wordfinder_ucs 1410
IG_PDF_doc_get_bookmark 1404
IG_PDF_doc_get_info 1405
IG_PDF_doc_get_layer 1406
IG_PDF_doc_get_layer_count 1406
IG_PDF_doc_print 1408
IG_PDF_doc_set_info 1407
IG_PDF_font_get_bbox 1418
IG_PDF_font_get_charset 1419
IG_PDF_font_get_cid_system_supplement. 1420
IG_PDF_font_get_cid_systeminfo. 1419
IG_PDF_font_get_descendant 1420
IG_PDF_font_get_encoding_index 1421
IG_PDF_font_get_encoding_name 1421
IG_PDF_font_get_font_matrix. 1422
IG_PDF_font_get_metrics 1422
IG_PDF_font_get_name. 1423
IG_PDF_font_get_subtype. 1423
IG_PDF_font_get_widths. 1424
IG_PDF_font_is_embedded 1424
IG_PDF_font_set_metrics 1425
IG_PDF_font_translate_string 1425
IG_PDF_font_translate_to_host. 1426
IG_PDF_font_translate_to_ucs 1427
IG_PDF_font_translate_widths 1428
IG_PDF_layer_create. 1430
IG_PDF_layer_delete. 1429
IG_PDF_layer_get_current_state 1429
IG_PDF_layer_get_initial_state 1429
IG_PDF_layer_get_intent. 1429
IG_PDF_layer_get_name 1429
IG_PDF_layer_get_usage_info 1429
IG_PDF_layer_has_usage_info 1429
IG_PDF_layer_remove. 1439
IG_PDF_layer_set_current_state 1429
IG_PDF_layer_set_initial_state 1429
IG_PDF_layer_set_intent 1429
IG_PDF_layer_set_name 1429
IG_PDF_layer_set_usage_info. 1429
IG_PDF_page_get_annotation_count. 1445
IG_PDF_page_get_content. 1441
IG_PDF_page_get_crop_box 1442
IG_PDF_page_get_rotation 1442
IG_PDF_page_make_color_separations . . . 1443
IG_PDF_page_release_content 1444
IG_PDF_page_set_content. 1444
IG_PDF_stream_close 1388
IG_PDF_stream_flush 1387
IG_PDF_stream_open_mem_for_read. 1384
IG_PDF_stream_read. 1386
IG_PDF_stream_read_CB_register 1385
IG_PDF_stream_write_CB_register. 1386
IG_PDF_style_delete 1456
IG_PDF_style_get_font_size 1456
IG_PDF_sysencoding_create_from_base_name . 1466
IG_PDF_sysencoding_create_from_cmap_name 1467
IG_PDF_sysencoding_create_from_code_page. . 1467
IG_PDF_sysencoding_get_writing_mode . . 1468
IG_PDF_sysencoding_is_identity 1469
IG_PDF_sysencoding_is_multibyte. 1469
IG_PDF_sysencoding_release 1470
IG_PDF_sysfont_enumerate. 1458
IG_PDF_sysfont_find. 1459
IG_PDF_sysfont_find_for_pdefont 1460
IG_PDF_sysfont_get_attrs 1461
IG_PDF_sysfont_get_cid_system_info 1462
IG_PDF_sysfont_get_create_flags. 1462
IG_PDF_sysfont_get_info 1463
IG_PDF_sysfont_get_name 1464
IG_PDF_sysfont_get_platform_data 1460
IG_PDF_sysfont_get_widths 1464
IG_PDF_sysfont_release 1465
IG_PDF_sysfont_release_platform_data . . . 1465

IG_PDF_TEXTENC_ASCII_85 . . 253, 261, 275
IG_PDF_TEXTENC_ASCII_HEX 253, 261, 275
IG_PDF_TEXTENC_NONE 253, 261, 275
IG_PDF_word_delete 1454
IG_PDF_word_get_char_offset 1449
IG_PDF_word_get_char_style 1450
IG_PDF_word_get_length 1451
IG_PDF_word_get_quad 1451
IG_PDF_word_get_quad_count 1452
IG_PDF_word_get_string. 1452
IG_PDF_word_get_style_transition 1453
IG_PDF_wordfinder_acquire_wordlist 1446
IG_PDF_wordfinder_delete 1448
IG_PDF_wordfinder_get_word 1447
IG_PDF_wordfinder_release_wordlist 1447
IG_pixel_bits_per_channel_get 1054
IG_pixel_channel_count_get 1054
IG_pixel_create. 1055
IG_pixel_data_pointer_get 1056
IG_pixel_delete. 1057
IG_pixel_value_get. 1058
IG_pixel_value_set 1059
IG_REGION_IS_NON_RECT. 604
IG_REGION_IS_NOT_AVAIL. 604
IG_REGION_IS_RECT 604
IG_runs_row_get 1061
IG_save_FD() 96, 206, 1064
IG_save_FD_CB_ex. 1066
IG_save_file() . . . 96, 135, 206, 1068, 1260, 1262
IG_save_mem() 206, 1072
IG_save_mem_CB_ex 1076
IG_save_tag_CB_register() 1078
IG_status_bar_CB_register() 1080, 1117
IG_thread_data_ID_associate() 1082
IG_thread_data_ID_get() 1083
IG_thread_image_lock(). 1084
IG_thread_image_unlock(). 1086
IG_thread_local_data_cleanup(). 1084
IG_tile_count_get(). 1087
IG_util_colorspace_alpha_count_get 1090
IG_util_colorspace_color_count_get 1090
IG_util_colorspace_contains_alpha 1091
IG_util_colorspace_contains_extra 1092
IG_util_colorspace_extra_count_get 1092
IG_util_colorspace_is_premultiplied 1093
IG_util_colorspace_is_valid 1094
IG_util_colorspace_value_to_ids 1095
IG_version_compile_date() 1096
IG_version_is_eval 1097
Imacon 473
Image Control for Strips 173
Image Correction 123
Image loading 41
Image Maintenance. 123
Image processing 42
Image Transformation 119
IMAGE_GEAR_LICENSE_DIR variable 67
ImageGear
    Architecture Diagram 84
    evaluation version 1097
    Licensing API 92
ImageGear component
    attaching 623
    checking 624
    function call 627
    list of loaded components 626
ImageGear Component Names. 187, 188
ImageGear Format Filter API. 243, 265
ImageGear library. 50, 205
ImageGear Pro
    Component Manager 187
INC_REND. 244
include files. 279
INDEPENDENT_PAGESIZE 244, 247, 250, 254, 258, ... 259, 260, 269, 271
Initializing the Component 242
Internet Services 52
Internet support. 45
IPTC Non-Image Data Structure 493
Item Count 217, 229, 1228, 1231, 1233, 1235, 1236, ... 1238, 1240

J

JOB_OPTIONS_FILE 254
JPEG 973, 1069
    Decimation 396
    Thumbnails 397
JPEG Compression
    Setting quality 206
JPEG compressions 396
JPEG Non-Image Data Structure 493
JPEG2K_ICC_PROFILE_APPLY. 204
JPEG2K_ICC_PROFILE_LOAD 204

K

Kodak 473
Konica Minolta 474

L

Largest Image Pixel Value 1262
Leaf 474

Leica 474
Length of a Data Field 227
LEVEL 254
Library Utility Functions 162
Linking 279
Linux 54
Loading DICOM ImagesLoading 205
Look Up Tables (LUTs) 81
Lookup tables 112
Lossless conversion of JPEG compressed image files 832
LPAFT_IG_ICC_GET_CB 1099
LPAFT_IG_METAD_ITEM_ADD_CB 141, 777, 778, ... 1099
LPAFT_IG_METAD_ITEM_GET_CB 141, 777, 778, ... 1101
LPAFT_IG_METAD_ITEM_SET_CB 141, 777, 778, ... 1103
LPFNART_GUI_TEXTREPLACEFUNC . . 1197
LPFNART_GUI_TOOLBARNOTIFYFUNC 1198
LPFNIG_ART_CREATENOTIFYFUNC . . 1199
LPFNIG_ART_DELETENOTIFYFUNC . . 1200
LPFNIG_ART_DRAWNOTIFYFUNC 1201
LPFNIG_ART_DRAWNOTIFYFUNC() . . 1206
LPFNIG_ART_MODIFYNOTIFYFUNC . . 1203
LPFNIG_ART_OVERRIDEDRAWNOTIFYFUNC 1205
LPFNIG_ART_PREMODIFYNOTIFYFUNC() . 1207
LPFNIG_ART_SELECTNOTIFYFUNC() 1208, 1210
LPFNIG_DIB_CREATE 970, 977, 978, 1105, 1114
LPFNIG_DIB_GET 755, 1106
LPFNIG_LOAD_DISP 165, 974, 1108
LPFNIG_MEM_ALLOC 1109
LPFNIG_MEM_FREE 1110
LPFNIG_MEM_REALLOC 1110
LPFNIG_MPCB_UPDATE 1111
LPFNIG_RASTER_GET 755
LPFNIG_RASTER_SET 970, 977, 978
LPFNIG_READ 756
LPFNIG_SEEK 755, 1115
LPFNIG_STATUS_BAR 1081
LPFNIG_TAG_GET 1078
LPFNIG_TAG_USER_GET 1078, 1125
LPFNIG_WRITE 755, 1127

M

Macros
    MED_DCM_DS_TAG_ELEMENT() 1251
    MED_DCM_DS_TAG_GROUP() 1251
    MED_DCM_DS_TAG_MAKE() 1252
Mandatory Data Elements 207, 230, 1215
Mark Attributes Structure 190, 191
Mark Blocks 191
Marks 199
    Arrow 199
    Attach A Note
        Font 200
        Set Background Color 200
    Attach-a-Note 200
    Bringing to the front 1175
    Burning into an image 194
    Checking selection status of 1181
    Checking show/hide status of 1182
    Clipboard OperationsMarksClipboard 192
    Components of
        Group Block 190
        Index Block 190
        Mark Attributes Structure 190
    Components ofMarkComponents 190
    Counting the number of marks in an image 1176
    Creating 1177
    Creating Using Low Level CallsCreating-MarkUsingLowLevelCalls 192
    Deleting 1179
    DeletingDeletingMarks 192
    Importing to an Image 195
    Index
        Getting or Setting for the current mark 1185
        Getting the index for the first mark 1180
        Getting the index for the next mark 1184
        Getting the index of the first mark in the selection 1189
        Getting the index of the next mark selected 1190
    Iterating through 194
    Modifying the attributes of 1183
    Polyline 202
    Querying the attributes of 1185
    Rotation 1186
    Saving to a FileSavingMarks 194
    Selecting/deselecting 1187
    Sending to the back 1191
    Setting the visibility (show/hide) of 1192
    Straight Line 202
    Typed Text 203
math library 280
MED_DCM_curr_data_get_string() 217
MED_DCM_display_color_limits() 221
MED_DCM_DS_bits_get() 1213
MED_DCM_DS_create() . 225, 1215, 1254, 1261

MED_DCM_DS_curr_data_get() 217, 229, 1216, 1217, ... 1250, 1256
MED_DCM_DS_curr_data_get_string() 217, 229, 1217
MED_DCM_DS_curr_data_set() . 217, 229, 1218
MED_DCM_DS_curr_index_get() . . . 229, 1219
MED_DCM_DS_curr_info_get() 217, 229, 1216, 1218, ... 1219
MED_DCM_DS_curr_remove() 228, 1220
MED_DCM_DS_DE_insert() 217, 225, 226, 1221
MED_DCM_DS_destroy(). 1222
MED_DCM_DS_exists() 1223, 1261
MED_DCM_DS_info_get() 224, 1224, 1236
MED_DCM_DS_is_empty() 1224
MED_DCM_DS_LUT_copy_get. 1225
MED_DCM_DS_LUT_exists 1226
MED_DCM_DS_LUT_update_from 1227
MED_DCM_DS_move_ascend(). 1228
MED_DCM_DS_move_descend(). 1229
MED_DCM_DS_move_find() . 228, 1230, 1250, 1256
MED_DCM_DS_move_find_first(). . . 228, 1232
MED_DCM_DS_move_first() 1234
MED_DCM_DS_move_index(). 1219, 1236
MED_DCM_DS_move_last() 1237
MED_DCM_DS_move_next(). 1239
MED_DCM_DS_move_prev(). 1241
MED_DCM_DS_orig_TS_get(). . 224, 225, 1243
MED_DCM_DS_part10_get() . . . 219, 225, 1244
MED_DCM_DS_part10_set() 220, 225, 1245, 1261
MED_DCM_DS_PixPadVal_get() 221, 1245, 1248
MED_DCM_DS_PixPadVal_set() 221, 1246, 1247
MED_DCM_DS_preamble_get(). 219, 225, 1248
MED_DCM_DS_preamble_set() . 219, 225, 1249
MED_DCM_DS_Rescale_get() 1250, 1274, 1276, 1283
MED_DCM_DS_TS_get(). 1253
MED_DCM_DS_TS_set() 1254
MED_DCM_DS_update_file 1254
MED_DCM_DS_Window_Level_get() 1250, 1255, 1274
MED_DCM_load_DICOM() 1256, 1258
MED_DCM_load_DICOM_FD() 1257
MED_DCM_save_DICOM() 206, 224, 1258, 1262
MED_DCM_save_DICOM_FD() 206, 1261
MED_DCM_util_data_to_string() 1263
MED_DCM_util_tag_info_add() . 226, 231, 1264
MED_DCM_util_tag_info_free(). 1265
MED_DCM_util_tag_info_get() . 217, 226, 231, 1265
MED_DCM_util_VR_info_mode() 217, 226, 1266
MED_DCM_util_VR_info_string() . . . 226, 1268
MED_display_color_create() . . . 221, 1269, 1273
MED_display_color_limits() 1271
MED_display_color_set(). 221, 1270, 1273
MED_display_contrast(). 1250, 1274, 1276, 1279
MED_display_contrast_auto() . . 221, 1275, 1280
MED_display_grayscale_LUT_build. 1277
MED_display_grayscale_LUT_build_auto . 1278
MED_IP_contrast(). 223, 1278
MED_IP_contrast_auto() 223, 1279
MED_IP_high_bit_transform() . 223, 1214, 1280
MED_IP_histo_clear() 223, 1281
MED_IP_histo_tabulate(). 1281, 1282
MED_IP_histo_tabulate(), ... 223
MED_IP_min_max() 223, 1248, 1283
MED_IP_normalize() 223, 1284
MED_IP_promote_to_16_gray() 223, 1285
MED_IP_reduce_depth_with_downshift(). . 223, 1214, ... 1280, 1285, 1287
MED_IP_reduce_depth_with_LUT(). 223, 1214, 1286
MED_IP_swap_bytes(). 223, 1287
MED_IP_window_level()
  incorporated functionality 1279
MED_modality_transform_apply. 1288
MED_PS_GSDF_LUT_init 1295
MED_VOI_window_init_from_min_max . . 1299
MED_VOI_window_max_get 1299
MED_VOI_window_min_get 1300
MedAPI.h 1270
Medical Constants
  DCM_PART10_ITEM_IMPLCLASSUID 230
  DCM_PART10_ITEM_IMPLVERNAME 230
  DCM_PART10_ITEM_MSSOPCLASSUID 230
  DCM_PART10_ITEM_MSSOPINSTUID 230
  DCM_PART10_ITEM_PREAMBLE 230
  DCM_PART10_ITEM_PRIVINFO 230
  DCM_PART10_ITEM_PRIVINFO_SIZE 230
  DCM_PART10_ITEM_PRIVINFOCRUID 230
  DCM_PART10_ITEM_SRCAETILE 230
  DCM_PART10_ITEM_TRANSSYNTAXUID 230
  DCM_PART10_ITEM_VERSION 230
  DCM_PSEUDOCOLOR_SCHEME_ 1270
  MED_DCM_MOVE_LEVEL_FIXED 1231, 1235, 1238, 1242
  MED_DCM_MOVE_LEVEL_FLOAT 1231, 1235, 1238, 1242
  MED_DCM_PLANAR_ 1260
  MED_DCM_TS_ 1215, 1243, 1260, 1261
  MED_DCM_TS_EXPLICIT_ 1253, 1254

MED_DCM_TS_IMPLICIT_ 1253, 1254
MED_DCM_VR_ 1263
Medical Formats
DICOM 3.0 Part 10-compliant 50
DICOM 3.0 Raw 50
Memory, saving an image to. 98
Message Box windows 281
Metadata global parameter 493
Metadata Item Name and ID constants
EXIF-JPEG, EXIF-FPX 526
EXIF-TIFF 518
IPTC 539
JPEG 512
PNG 549
TIFF 504
Metadata Structure “ValueType” and “Value” 493
Metafile Formats
support for 288
Meta-Info Header
Setting Item Data for 225
MONOCHROME1 226
MONOCHROME2 226

N

Named Blocks. 191
Creating 1167, 1169
Deleting 1170
Getting the block data 1174
Getting the first block name 1171
Getting the next block name 1173
Nikon 474
Ñlipboard
merging in image 605
NOCIE 269
Non-Image Data Structure
CIFF 526
EXIF-TIFF 512
IPTC 531
JPEG 504
PNG 539
TIFF 497
Non-image Data Structure
EXIF-Flashpix extensions 523
EXIF-TIFF 519
non-rectangular region of interest. 601
Normalizing 1284
Notes(ART)
Attach-A-Note 200
NOTRANSPARENCY. 270

O

Olympus 475

P

Padding 173
page height 246, 256, 258, 269, 271
page size 246, 256, 269
page width. 246, 247, 256, 258, 269, 271
PAGE_HEIGHT 244, 246, 254, 258, 270
PAGE_WIDTH. 244, 247, 254, 258, 271
PALETTE COLOR 226
Palette Handling 108
Palettes 160
Panasonic 475
Part 10 Header 212, 1243, 1245, 1261
Access 219
Creating 230
Data Setting 224
Getting Data from 225, 1244
Populating 231
Preamble setting 225
Saving with Image 1262
Setting Item Data of 1245
Part 10 meta-info Header 225
Part 10 Template. 216
Part 3 of the DICOM specification. 207
Part 5 of the DICOM specification. 226
Part 6 of the DICOM specification. 1215
PASSWORD. 244
PDF and PS fonts 242
PDF Component 263
PDF Component attaching 237
PDF file loading
speeding up 269, 270
PDF Filter Control Parameters 244, 266
PDF Format Filter. 243, 265
Save Compressions 243
Supported Features 243, 265
PDF Frmat Filter
Save Compressions 266
PDF Layers. 239
PDF Plug-in 58
PDF Support 264
Pentax 475
Photometric Interpretation 224, 1215
Pixel
allocating space for 125
Pixel access. 43
Pixel Data 224, 1215
Pixel Padding Value 1246, 1247

about 220, 1247
Retrieving 1246
Setting 1247
Pixel Representation 224, 1215
pixel's value
setting 639
Pixels
accessing 124
getting and setting 125
getting and setting linear groups of 126
obtaining info about 637
rectangular areas of 127
Planar Configuration 224, 1215, 1260, 1262
PNG Non-Image Data Structure 493
Pointers
Getting the pointer to a HIGEAR's Palette IG_image_DIB_palette_pntr_get() 849
PolylineSee Marks 202
poorly 1287
Poorly constructed images
Fixing 1287
PPM
convert to DPI 629
Preparing Your Application for the End User 281
Preserve Black 935, 938
Preserve White 935, 938
PRINT_DEPTH 244
PRINT_RESOLUTION_X 244
PRINT_RESOLUTION_Y 244
Printing 42
Printing Images 130
Pseudocolor
Applying to the darkest and lightest pixels 1272
Pseudocolor schemes 1270
Applying to 8-16 bit gray-scale images 1272, 1273

Q

Quantization 301

R

raster image 81
raster object 264
rasterization 264
READ_PREVIEW 254
Reading Data From Data Elements 217
rectangular region
number of bytes 641
Reducing bit depth
Using default display LUT 1286
Reduction 118
Removing Your Debugging Error Messages . 281
Rescale Intercept. 1250, 1274, 1276, 1279, 1280, 1283
Rescale Slope 221, 223, 1250, 1274, 1276, 1279, 1280, ... 1283
RESOLUTION_X 244, 250, 254, 260, 272
RESOLUTION_Y 244, 250, 254, 260, 272
Restriction flags 226, 1267, 1268
RGB 226
converting to hue 942
Pseudocoloring 221
threshold settings 1272
ROI Processing 43
Rotating
Actual bitmap data 858
Rows 224, 1215
Run 1061

S

Sales Department
Phone Number 48
SAMPLES 65
Samples Per Pixel 224, 1214, 1215
SAVE_FLAGS 244
SAVE_MAJOR 244
SAVE_MINOR 244
Saving a non-DICOM Image as a DICOM image 207
Saving DICOM
Using DICOM Specific Functions 206
Using File Descriptor 206
Using ImageGear Functions 207
Saving marksSavingMarks 194
Saving Options
DCM_CONTROL_SAVE_INCLUDE_GROUP_LENGTHS 213
DCM_CONTROL_SAVE_INCLUDE_LARGEST_PIXEL 214
DCM_CONTROL_SAVE_JPEG_QUALITY 213
DCM_CONTROL_SAVE_PLANAR_CONFIG_TYPE 214
DCM_CONTROL_SAVE_SYNTAX 215
Scanning Options 493
Scroll Bars 74
Sequence Delimiter 219
Sequence of Items (SQ) 227
Sigma 475
Signed images 223, 1276, 1280

Checking for 1282
Converting signed to unsigned 1284
Status 1283
SMOOTH_FLAGS 244, 254
SOFTWARE LICENSE AGREEMENT 581
Solaris 54
Sony 475
Special effect. 43
Special Effects 121
Straight LineSee Marks 202
Stripped Images 171
subbands 302
Support and Maintenance Policy 588
Support and maintenance services
Preferred Level 589
Premier Level 591
Primary Level 588
symmetric extension 302

T

Table 10 886
Tag Values 171
Tags
Adding new 1264
Creating 1252
DIMSE 210
Getting Information about 1266
Searching for 1231
Searching for the first Tag 1233
Setting the Tag value 226
Setting the VR 226
User-defined 231
Emptying table of user-defined Tags 1265
VR of CS 226
Window Center 1256
Window Width 1256
Targa 836
text encoding scheme 244, 266
text extraction 51, 264
TEXT_ALPHA. 253, 261, 273
TEXT_ENCODING 244, 253, 254, 261, 275
thread safety 130
Thumbnails
loading 988
TIFF format. 1078
TIFF Non-Image Data Structure. 493
TIFF Storage 179
TIFF Tags 1078
TIFF-CMYK images. 621
Tiled Images 173
Tiling. 301
Toolkit Libraries 280
Transfer Syntax. 1257
Getting the original Transfer Syntax 225, 1253
Setting 224
Setting for loading 211
Setting for saving 215, 1254
Transform
Discrete Fourier 302
Discrete Wavelet 301, 302
irreversible 302
reversible 302
Transitions 44
Transparency. 110
transparency
disabling 270
Transparency and Background 110
Tutorial
converting image 69
converting to DDB format 69
loading an image 69
scrollbars 69
tutorial. 69
TYPE 254
Typed TextSee Marks. 203
Types of DEs 230

U

Undo
Mark operations 1141
Uninstalling ImageGear 65
UNIX-specific functions. 69
USE_CROP_BOX 244, 254
User-defined Tags. 231
Utility Functions 158

V

Value Lengths (VL) 1216
Value Multiplicity (VM). . 226, 1218, 1264, 1266
vector data. 264
Vector Images. 83
VGA display 272, 273
VL 217, 1228, 1231, 1233, 1236, 1238, 1240, 1241
VM 217, 231, 1264
VR. 217, 231

W

WIDTH. 254
Window Center . . . 1274, 1275, 1276, 1279, 1280
Returning the Value of 1256

Window Width . . . 1274, 1275, 1276, 1279, 1280
Returning the Value of 1256
Writing Data to Data Elements 217

X

X Windows System 280

Y

YBR_FULL 226
YBR_FULL_422 226
YBR_PARTIAL_422 226